ISSN 0265-2919

90p 91

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 Sing $4.50

CONTENTS

APPLICATION

DP OPPORTUNITIES The variety of jobs available in data processing ranges from keyboard operation to systems analysis — 1810

HARDWARE

INTUITION AND IMAGINATION Is the anticipation surrounding the Commodore Amiga justified by its performance? — 1801

SOFTWARE

PARENTAL INFLUENCE Based on a system of hierarchical directories, Unix provides a powerful user environment — 1808

IN THE PIPELINE It is essential when using MS-DOS that you are constantly aware of the current sub-directory — 1813

A RARE BEAST The first games program with a built-in safeguard against piracy — 1820

COMPUTER SCIENCE

POINTING THE WAY Pointers are a special C variable type — 1816

JARGON

FROM TRUTH TABLE TO TWO'S COMPLEMENT A weekly glossary of computing terms — 1807

PROGRAMMING PROJECTS

PLACE YOUR BETS The betting routines complete our pontoon program — 1804

MACHINE CODE

GETTING INTO THE SUBROUTINE We look at the use of subroutine calls on the Motorola 68000 — 1818

Next Week

- **Recent evidence indicates that communications is becoming increasingly popular among micro enthusiasts. We take a look at some of the most popular modems available.**
- **Storing characters in eight-bit ASCII format often uses large quantities of memory. A new series looks at text compression techniques.**
- **We conclude our investigation of the MS-DOS operating system.**

QUIZ

1) On what other operating system is AmigaDOS based?
2) Where does the Mikro-Plus EPROM reside in memory, what memory advantage does it produce and why?
3) What is a 'pipe'?

Answers To Last Week's Quiz

1) In order to use the PCW 8256 with a different printer, the user will have to obtain the necessary interface, and print the files from CP/M rather than from LocoScript.
2) The types of variables provided by C are: automatic, external, register and static.
3) TOPs, an acronym for Training Opportunities Programmes, are government schemes intended to retrain workers.

Coming Up

- A discussion of some of the developments expected in the microcomputer industry in forthcoming years.
- A look at the things to consider when choosing a computer language.

FACT SHEET A fact sheet to complement our Machine Code series on the Motorola 68000 — INSIDE BACK COVER

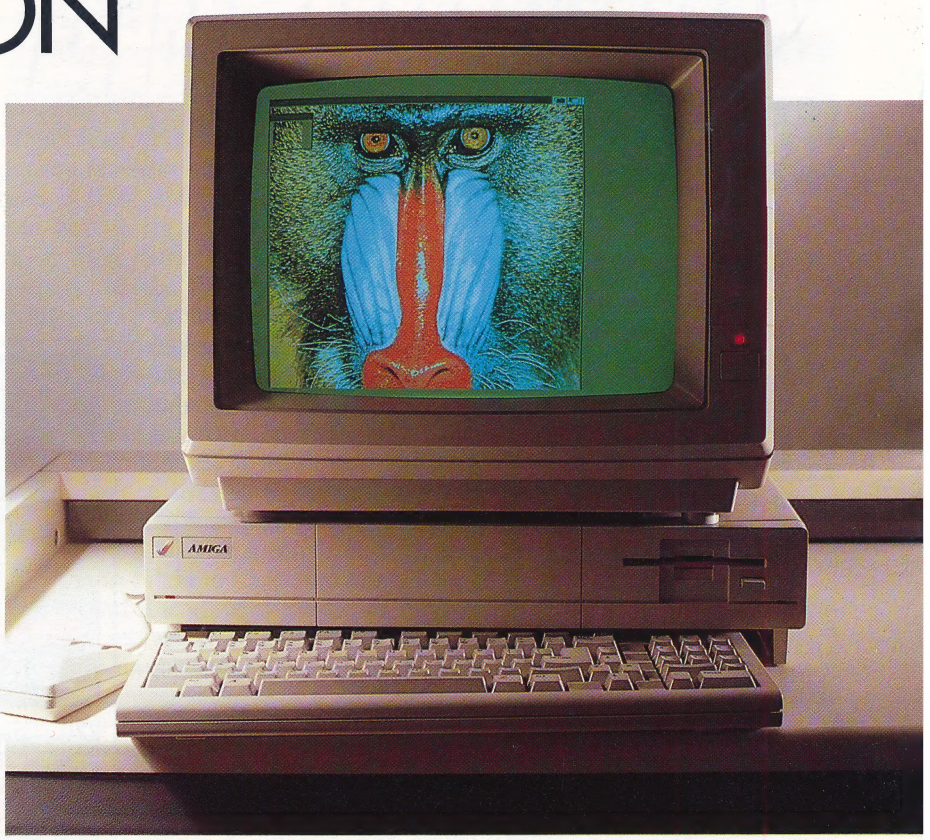
Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editor** Jon Kaye; **Contributors** Mike Curtis, Steve Colwill, Steve Malone, Nigel Cross, David Fensome, Max Hotopf, Peter Shaw; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, Locked Bag No. 1, Cremorne, NSW 2090. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited. Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH SHOT AT COMMODORE U.K. CORBY

INTUITION AND IMAGINATION

It's rare that the arrival of a new machine is preceded with enormous anticipation by the microcomputer industry. But 'leaks' from various quarters ensured that the Commodore Amiga created a great deal of excitement because of its innovative design and exceptional performance.



MARCUS WILSON-SMITH

Evolutionary Leap
The Commodore Amiga represents a dramatic new development in the evolution of the microcomputer. The extensive use of custom-built chips, utilising 'blitter' technology, greatly enhances the sound and graphics capabilities of the machine, while freeing the CPU to manage other processes

The Amiga was originally developed by the Amiga Corporation in the USA and was later acquired by Commodore. Although the machine has appeared in the USA under the Commodore umbrella, none of its badges mention the company name. Because the computer is aimed at the business market or the serious user, and Commodore is perceived in the USA as a small games machine manufacturer (based largely on the success of the Commodore 64), the company decided to market the machine with a completely fresh image.

Housed in a single casing with a built-in disk drive, the Commodore Amiga looks very business-like. The keyboard, which is separate from the computer and connected by a generous length of cable, has legs on the bottom that can be raised to angle the keys for more comfortable typing. Although the feel of the Amiga keyboard is not the very best around, it is certainly adequate for most applications. To the bottom-right of the typewriter keys is a cursor cluster that can be used to replicate the functions of the hand-held mouse.

The Amiga comes fitted with one double-sided $3\frac{1}{2}$in disk drive, into which the manufacturers have managed to cram up to 880 Kbytes of data. This gives the Amiga access to more data on a single drive than many computers have on twin drives.

The front panel to the left of the drive reveals an expansion slot when removed. Fitted with 256 Kbytes of memory as standard, the Amiga can accommodate an additional memory module that simply slots into the front panel, upgrading the machine to 512 Kbytes. To expand the Amiga further towards the maximum configuration of eight Mbytes, there is a second slot provided on the right-hand side of the computer casing through which the additional memory can be interfaced. Also provided on this side of the machine are a pair of nine-pin D connectors provided for joysticks or, more relevantly, the mouse controller.

The rear of the computer houses the peripheral interface connections. From the left these are the keyboard port, Centronics printer port, and an interface for a second disk drive. The Amiga also possesses an RS232C connection for modem and other serial peripherals and a pair of phono sockets. These can be plugged directly into the dedicated monitor provided for the Amiga or, more profitably, into a hi-fi system where the quality of the computer's sound can be fully appreciated.

The remaining three ports are reserved for video functions, including a 23-way connector for RGB monitors and a socket for composite video. Also available is a 'video in' socket that allows the Amiga to input video pictures from a VCR and display computer–generated images on top of the pictures — rather like the Pioneer PX-7 system (see page 1589). Future hardware add-ons for the Amiga are likely to include a 'frame grabber' that digitises a frame from a video source, which can then be reflected, rotated or otherwise manipulated under software control.

At the heart of the Amiga is Motorola's 68000 processor (found in the Apple Macintosh and Atari 520ST), but the feature that gives the Amiga its outstanding performance is the interplay of three custom chips with each other and with the 68000. Named Agnus, Daphne and Portia, the chips can control video display, sprites and disk I/O more or less autonomously, freeing the 68000 to perform processing applications at full speed. Agnus, for example, has its own co-processor and 'bit image manipulator' (blitter) that allows it to move a million pixels per second. Line-drawing

All Keyed Up
The keyboard provided with the Amiga is among the best currently on the market. In keeping with modern trends, the keyboard includes a numeric keypad, 10 programmable function keys and a number of other keys that can be used for control functions

Metacomco In Amiga

The Bristol-based software house Metacomco was formed in 1981 to develop systems software for 16- and 32-bit computers. Its Personal BASIC, developed for Digital Research, has become a standard on computers that run CP/M-86. More recently, Metacomco has concentrated its activities on writing 68000 software and has produced versions of PASCAL, LISP and C for the Sinclair QL and Atari 520ST, as well as 68000 assemblers and software development packages.

Metacomco was approached to write AmigaDOS for the fledgling Amiga when another system house, initially contracted to develop an operating system, failed to deliver the goods. AmigaDOS is based on a multi-tasking, networking system known as Tripos, which began life several years previously as a PhD project at Cambridge University. Dr Tim King, head

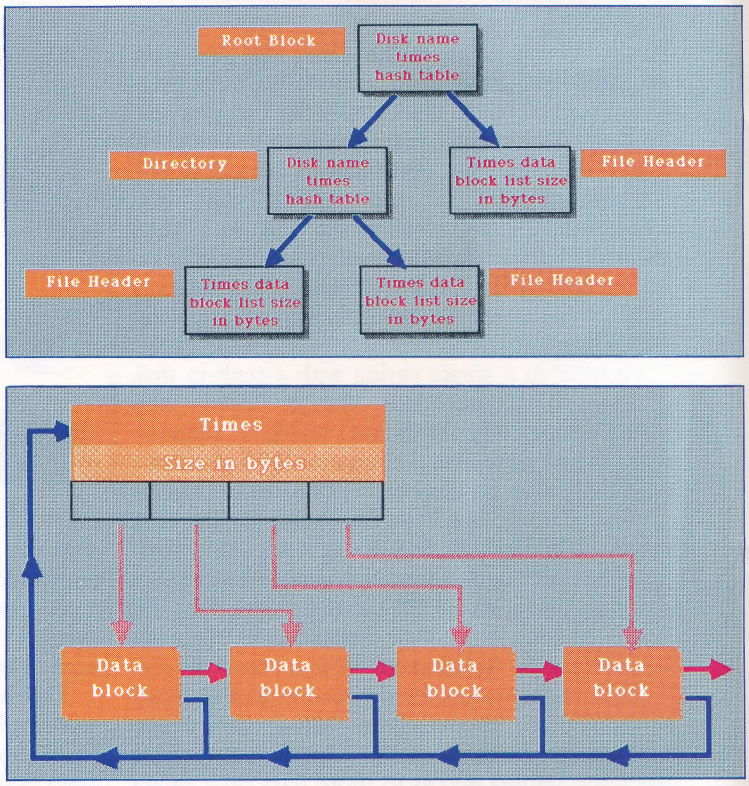
and shape-filling logic also form part of Agnus's circuitry. All of this means that the Amiga has the capability to move, alter and fill shapes so fast that it creates the illusion of being instantaneous. In addition, the blitter is used to transfer disk data between the disk buffers and memory.

Two types of sprites are available. The first, known as 'Vsprites', are controlled from hardware, allowing them to move around the screen at high speed. The second type of sprite is software controlled using the blitter. These 'blitter objects' (bobs) allow more complex shapes and colouring than Vsprites can achieve. The sum of all these features is that the Amiga can produce graphics of a speed and quality previously available only on dedicated arcade machines.

The sound capabilities are also extremely impressive — matching the quality of many commercial synthesisers. The Amiga is able to reproduce digitally sampled real sounds that can then be manipulated to any given pitch, complete with stereo imaging. Also included as standard is speech synthesis, which can talk to you in male or female voices, add inflections and process written text. Word processors that read your work back to you or read messages from your electronic mailbox are just two applications that spring to mind. The BASIC provided with the system — AmigaBASIC — supports all these features and enables impressive graphics to be produced from a few lines of program. (Speech synthesis is supported by BASIC commands that translate a text string into a string of phonemes, which are then 'spoken'.)

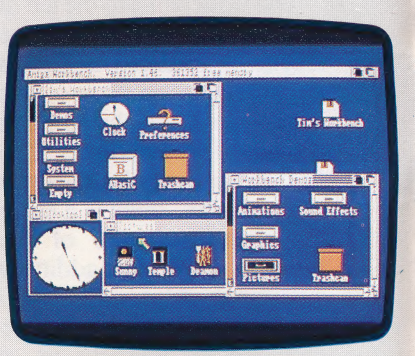
The Amiga presents a friendly WIMP-type environment to the user, called Workbench, in the style of the Macintosh and GEM-based machines. Files can be loaded by pointing to appropriate icons and clicking a button on the mouse. Beneath this is the operating system proper, known as AmigaDOS, which makes the machine truly multi-tasking. AmigaDOS, in turn, makes calls to Intuition, the part of the firmware that looks after window and mouse control.

In practice, the Amiga's multi-tasking capabilities allow you to have several applications running simultaneously yet independently. This is likely to be a major selling point to the business



Pictures At An Exhibition

The quality of the Commodore Amiga's graphics sets it far above other machines in its price range. Complex animation and high resolution graphics with subtle colouring make the Amiga an arcade machine par excellence. Its Macintosh-like WIMP operating environment is likely to attract many non-specialist users such as graphic designers and artists



The Amiga is operated through an iconic interface, known as Workbench. The familiar windows, disk and trash can icons are present

of Metacomco's R&D department, was given the task of customising Tripos into a functional operating system for the Amiga in a single month.

Metacomco then became closely involved with the Amiga project, producing AmigaBASIC — the BASIC interpreter bundled with the Amiga — PASCAL and LISP. It has also written a macro assembler and development systems that run under Unix and MS-DOS.

Although AmigaDOS does not currently support networking, Tripos is fundamentally a networking operating system — plans look to future versions of AmigaDOS to allow the networking of up to 255 Amigas. Tim King sees this type of system as an alternative to mainframe computing facilities. The network is flexible enough to accept extra machines when necessary, each capable of using another station's peripherals or accessing a centralised file server equipped with a hard disk

Backwards And Forwards

The file system used by AmigaDOS is unusual. Traditional disk-filing structures have a directory track that contains the names of all the files on the disk and pointers to the first data blocks. AmigaDOS's filing system is based on a tree structure with a complex set of forward and backward pointers linking the blocks on the disk. There is no directory track as such, but a 'root' block with pointers to file headers or other directories.

The file header contains a series of pointers to each data block in the file and other information about the file — such as its size in bytes and the time it was last accessed. The data blocks in the file are also chained together by a series of forward pointers and each data block also points back to the file header to which it belongs.

This complex system of pointers has one very important implication. If a disk is corrupted then, given only a single 'good' block on the disk, the entire disk filing structure can be recreated by following and remaking pointer links between blocks. The integrity of data is of great importance, particularly to business users, and the Amiga's ability to salvage corrupted disks is likely to add to its appeal

sector since it is the first micro in its price bracket to provide such facilities. Because the processing capabilities of the machine are shared out on a 'time-slice' basis (each application receiving short intervals of attention in turn) there is a commensurate loss of speed as more applications are multi-tasked together.

IBM COMPATIBILITY

A major problem for all computer manufacturers aiming at the business market in the USA is the dominance of IBM and the reluctance of business computer users to try new machines (hence the plethora of IBM clones). Although the Amiga easily outperforms the IBM PC at about half the price, Commodore has further tried to ensure the success of the Amiga by producing a 5¼in disk drive and emulation software option that configures the machine to an IBM-compatible. Commodore claims that this enables the Amiga to run packages such as Lotus 1-2-3. The emulation technique, in essence, translates 8088 op-codes to 68000 op-codes. Obviously, this translation process is bound to slow the Amiga down, but there are plans to produce a hardware emulation board that should alleviate this problem.

The Amiga sets new standards in terms of speed, graphics and sound, and the company's claim that its innovative approach will win it completely new markets may well be justified. A new computer, however, ultimately stands or falls by the amount of high-quality software available for it. On its launch in the USA, the Amiga had only eight major software packages written for it. There are signs, however, that a number of software houses are producing programs for the machine, including adaptations of some of the most popular Macintosh packages.

The Amiga could be classified as an extremely impressive but expensive games machine, or a cheap and powerful business machine. Thus in some ways the machine might suffer from an identity crisis that can only be resolved by the market response.

Commodore Amiga

PRICE

Approximately £1,500

DIMENSIONS

444 × 300 × 120mm

CPU

Motorola 68000, running at 8 MHz

MEMORY

256 Kbytes, expandable to 8 Mbytes

SCREEN

The Amiga has a maximum text resolution of 80 × 25 characters. The four graphics modes available range from a 320 × 200 pixel, 32-colour mode, up to 640 × 400 pixel 16-colour mode

INTERFACES

Two mouse ports, expansion bus, RS232 interface, Centronics interface, second disk drive port, RGB and composite video ports

KEYBOARD

82 keys, including 10 function keys and numeric keypad

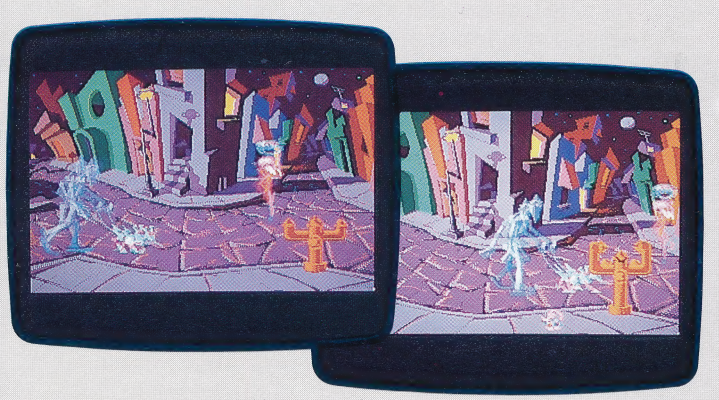
STRENGTHS

The Amiga allows true multi-tasking, and the capabilities of the sound and graphics are breathtaking

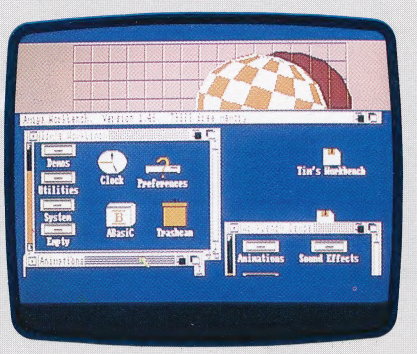
WEAKNESSES

Much of the software promised for the machine has not yet materialised. Furthermore, Commodore have not decided on a target market for the machine. Yet public perception, and hence the success, of the Amiga may depend on the marketing

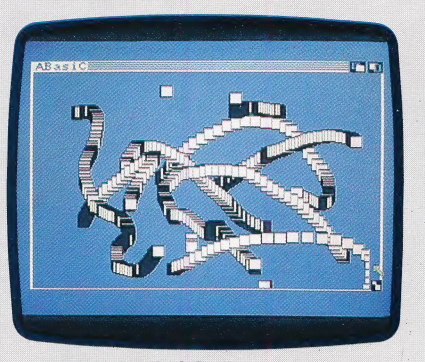
MARCUS WILSON-SMITH



The moving characters in this cartoon tableau are created using blitter objects, software 'sprites' that allow complex shape and colour definitions. Blitter objects include proximity detection. Here, the fire hydrant figure rises up to fend off the approaching dog



The Amiga is a multi-tasking micro as demonstrated here. A bouncing ball demonstration and Workbench can be seen running together



Although AmigaBASIC is by no means an excellent implementation of the language, it does support most features of the machine. A five-line BASIC program was sufficient to produce this doodle display with the mouse

PLACE YOUR BETS

We conclude our pontoon programming project by adding betting routines to the game and including a title screen. The betting routines allow you to place the initial bets, and double a bet on a promising hand.

Betting on the turn of the cards is an integral part of the game of pontoon. The program gives you an initial stake of £10,000. The player's remaining capital is held throughout the game in the variable SK. Various methods of betting in pontoon exist. Our version of the program will adopt the following system:

- The player must bet £50 at the start of each round to enter the game.
- This initial bet is made automatically by the program.

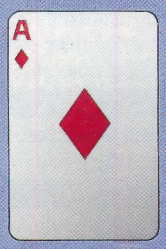
Having been dealt one card, the player can then 'buy' a second card by making an additional bet of up to £1,000. A shrewd player will recognise that some cards (for example, an ace) are more promising than others and bet accordingly.

Additional cards can then be dealt to the player until he sticks or busts. If the player thinks that he has an extremely promising hand, or if he underbets on his first card, then he can opt to double his bet and be dealt one more card.

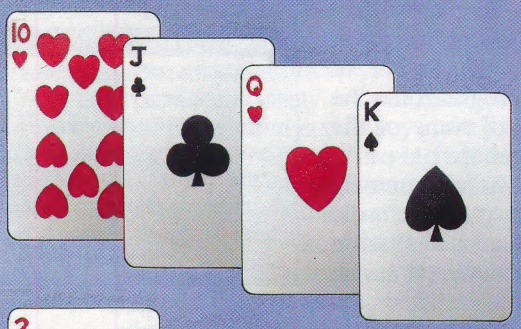
The player wins the round if he has the highest score after the computer (the 'banker') has played out its hand. In this case his bet is returned to him and an additional amount equal to his bet is added to his capital. Otherwise, he loses the money staked.

Betting Strategies

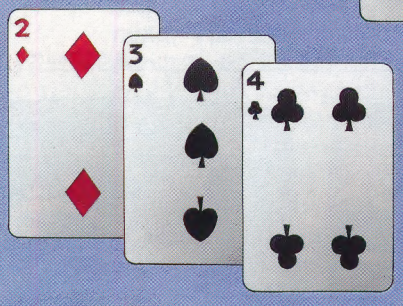
As with all betting, successful strategies are based on the principle 'win heavy and lose light'. The crucial phase of the betting in our version of pontoon is when the player buys a second card. Below we outline some simple suggestions to improve your performance



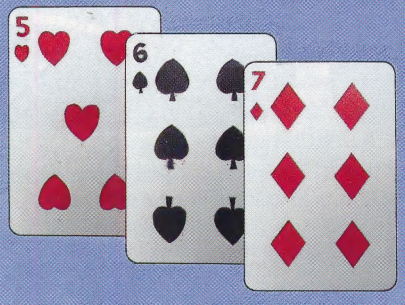
Worth Very High Stake
- Up to 16 cards in pack that give royal pontoon/pontoon
- Ace counts as 1 or 11 giving flexibility as hand unfolds
- Possible five-card trick



Worth High Stake
- Subsequent 2, 3 or 4 allow cards to be burned
- High second card (9, 10, J, Q, K) allows a good stick
- Subsequent ace gives pontoon/royal pontoon
- Drawback in that a subsequent 'middle order' card (5 or 6) makes third card likely to bust hand



Worth Medium Stake
- Low cards give chance of five-card trick
- Possibility of burn if second card is high



Low Stake Only
- Middle order cards can give problems if 9, 10, J, Q, K dealt as second card

KEVIN JONES

THE BETTING BOX

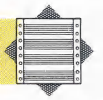
Throughout the game, the current bet and remaining stake levels are displayed on the screen. Headings for this box are printed by the routine at line 4200, which is called from the 'initialise game' routine at line 625. This line also resets a variable, BT, which is used to keep track of the bet made during the current round.

The 'print stake' routine, which starts at line 4300, is a general-purpose routine that can be called from several points of the main program. Additional bets are made by setting the variable SB before calling the routine. Under normal circumstances, SB is added to the cumulative bet, BT, and subtracted from the remaining stake, SK. However, as the routine is general it needs to make several checks on the validity of the bet to be made. If the player has less than £50 remaining, and the initial automatic bet is made (indicated by the flag BG=1) the routine will print a message stating that insufficient funds remain and give the player the option of restarting the game from the beginning.

If the player wishes to double his stake, but does not have sufficient capital, then a message must be printed and the bet disallowed. This test is made by subtracting the projected bet and seeing if a negative remaining stake, SK, is left. If so, then the bet is added back and the routine is exited.

The third unusual circumstance is if the player attempts to buy a second card by betting more money than he possesses. In this case, the routine will print an appropriate message and reduce the additional bet to the amount of capital left to the player.

The routine then goes on to erase the previously printed bet and remaining stake values by overprinting a number of space characters, before printing the new values.



USING THE BETTING ROUTINE

The initial bet of £50 is made automatically at the start of each round by line 72. Lines 73 to 80 allow the player to buy a second card, checking the amount entered to ensure that it does not exceed the £1,000 limit.

As the player twists to receive more cards in the routine beginning at line 2700, he can choose to double on his last card. The double routine itself is at line 2900 and calls the 'print bet' routine. If the player cannot afford to double (indicated by CA=1) then the routine is exited and the program reverts to the normal twist/stick routine. Otherwise, the player is dealt a final card, the hand is evaluated and the routine terminated.

In the previous instalment of the project we developed the routines that decide which hand has won and then print a 'win or lose' message, as appropriate. All that remains is to add a line that will add the amount betted to the remaining stake

Betting Routines

BBC Micro

```
   30 GOSUB 4000
   72 BG=1:SB=IS:GOSUB 4300:BG=0
   73GOSUB 700:PRINT"BUY A CARD (Y/N) ";
   74 AN$=GET$
   75 IF AN$<>CHR$(13) THEN PRINT AN$
   77 IF AN$<>"Y" THEN 85
   78 AN$="":GOSUB 700:INPUT "YOUR BET (
MAX £1000)"AN$
   79 IF VAL(AN$)>1000 THEN 78
   80 SB=VAL(AN$):GOSUB 4300
  210 GOSUB 700:PRINT"YOU WIN £";BT
  220 SB=-2*BT:BT=-SB:GOSUB 4300
  565 SK=10000:IS=50
  610 HP(1)=1:HP(2)=1
  625 BT=0:GOSUB 4200
 2740 IF AN$="D" THEN GOSUB 2900:IF CA=0
 THEN RETURN
 2900 REM **** DOUBLE ****
 2910 DB=1:SB=BT:GOSUB 4300
 2915 IF CA=1 THEN DB=0:RETURN
 2920 FL=0:PL=1:GOSUB 1300
 2930 GOSUB 800
 2940 DB=0:RETURN
 4000 REM
 4010 CLS
 4020 TX=13:TY=3:GOSUB 900:PRINT"BBC/ELE
CTRON"
 4030 PRINT TAB(TX+3);"PONTOON"
 4040 PRINT TAB(TX+5);"BY"
 4050 PRINT TAB(TX-7);"PETE SHAW & STEVE
 COLWILL"
 4060 TX=9:TY=8:GOSUB 900:PRINT"YOUR STA
KE IS £";SK
 4070 TX=10:TY=12:GOSUB 900:PRINT"PRESS
ANY KEY TO PLAY"
 4080 A$=GET$
 4090 RETURN
 4200 REM
 4210 COLOUR 4:TX=24:TY=18:GOSUB 900:PRI
NT"YOUR BET"
 4220 TX=24:TY=20:GOSUB 900:PRINT"REMAIN
ING STAKE"
 4230 RETURN
 4300 REM
 4302 CA=0:REM CAN'T AFFORD TO DOUBLE
 4305 IF SK>=50 OR BG=0 THEN 4310
 4306 AN$="":GOSUB 700:INPUT"OUT OF CASH
! PLAY AGAIN(Y/N) ";AN$
 4307 IF AN$="Y" THEN RUN
 4308 END
 4310 SK=SK-SB:BT=BT+SB
 4315 IF DB=1 AND SK<0 THEN GOSUB 700:PR
INT"YOU CAN'T AFFORD IT!"
 4317 IF DB=1 AND SK<0 THEN BT=BT-SB:CA=
1:RETURN
 4320 IF SK<0 THEN BT=SK+SB:SK=0:GOSUB 7
00:PRINT"YOU CAN ONLY AFFORD £";BT
 4340COLOUR 1:TX=24:TY=19:GOSUB 900:PRIN
T LEFT$(SP$,15)
 4345 TX=24:TY=19:GOSUB 900:PRINT "£";BT

 4350 TX=24:TY=21:GOSUB 900:PRINT LEFT$(
SP$,15)
 4355 TX=24:TY=21:GOSUB 900:PRINT "£";SK

 4360 RETURN
```

Amstrad CPC Range

```
30 GOSUB 4000:REM title screen
72 bg=1:sb=is:GOSUB 4300:bg=0:REM print
bet
73 GOSUB 700:PRINT "buy a card (y/n)";
74 an$="":WHILE an$="": an$=INKEY$:WEND
75 IF an$<>CHR$(13) THEN PRINT an$
77 IF an$<>"y" THEN 85
78 an$="":GOSUB 700:INPUT"your bet (max
£1000)";an$
79 IF VAL(an$)>1000 THEN 78
80 sb=VAL(an$):GOSUB 4300:REM print bet
210 GOSUB 700:PEN black:PRINT"you win £"
;:PEN white:PRINT bt
220 sb=-2*bt:bt=-sb:GOSUB 4300:REM print
 bet
565 sk=10000:is=50:REM stakes
610 hp(1)=1:hp(2)=1
625 bt=0:sb=0:GOSUB 4200:REM print bet b
ox
2740 IF an$="d" THEN GOSUB 2900:IF ca=0
THEN RETURN
2900 REM **** double ****
2910 db=1:sb=bt:GOSUB 4300:REM print bet
2915 IF ca=1 THEN db=0:RETURN
2920 fl=0:pl=1:GOSUB 1300:REM deal
2930 GOSUB 800:REM evaluate
2940 db=0:RETURN
4000 REM **** title etc ****
4010 CLS
4020 tx=11:ty=3:GOSUB 900:PEN red:PRINT"
Amstrad CPC Range"
4030 PRINT TAB(tx+6)"Pontoon"
4040 PRINT TAB(tx+8)"By"
4050 PRINT TAB(tx+3)"Steve Colwill"
4060 tx=9:ty=8:GOSUB 900:PEN black:PRINT
"Your stake is £";sk
4070 tx=10:ty=12:GOSUB 900:PEN white:PRI
NT"Press a key to play"
4080 WHILE INKEY$="":WEND
4090 RETURN
4200 REM **** betting box ****
4210 tx=24:ty=18:GOSUB 900:PEN red:PRINT
"Your bet"
4220 tx=24:ty=20:GOSUB 900:PRINT"Remaini
ng stake"
4230 RETURN
4300 REM **** print stake ****
4302 ca=0:REM can't afford to double fla
g
4305 IF sk>=50 OR bg=0 THEN 4310
4306 an$="":GOSUB 700:PEN red:INPUT"Out
of cash! Play again (y/n)";an$
4307 IF an$="y" then run
4308 END
4310 sk=sk-sb:bt=bt+sb
4315 IF db=1 AND sk<0 THEN GOSUB 700:PEN
 red:PRINT"You can't afford it!":bt=bt-s
b:sk=sk+sb:ca=1:RETURN
4320 IF sk<0 THEN bt=sk+sb:sk=0:GOSUB 70
0:PEN red:PRINT"You can only afford £";b
t
4340 tx=24:ty=19:GOSUB 900:PRINT SPACE$(
15)
4345 tx=24:ty=19:GOSUB 900:PEN white:PRI
NT"£";bt
4350 tx=24:ty=21:GOSUB 900:PRINT SPACE$(
15)
4355 tx=24:ty=21:GOSUB 900:PRINT"£";sk
4360 RETURN
```

total in the event that the player wins the hand. This is done by inserting line 220 into the main program loop.

The program listings are now complete for each of the four machines and will allow the player to play the full game against the computer. The program will automatically shuffle at the beginning of the game and can be made to automatically reshuffle the deck after any round by pressing SHIFT/S (SYMBOL SHIFT/S on the Spectrum).

As a further project, you might wish to enhance the card display and dealing routines given and create a pontoon game for more than one player.

Sinclair Spectrum

```
  30 GO SUB 4000: REM TITLE SCREEN
  72 LET BG=1: LET SB=IS: GO SUB 4300: L
ET BG=0: REM PRINT BET
  73 LET A$="": GO SUB 700: PRINT  INK 2
;"BUY A CARD (Y/N) ";
  74 LET A$=INKEY$: IF A$="" THEN  GO TO
 74
  75 IF A$<>CHR$ 13 THEN  PRINT A$
  77 IF A$<>"Y" THEN  GO TO 85
  78 LET A$="": GO SUB 700: INPUT "YOUR
BET (MAX £1000)"; LINE A$
  79 IF VAL A$>1000 THEN  GO TO 78
  80 LET SB=VAL A$: GO SUB 4300: REM PRI
NT BET
 210 GO SUB 700: PRINT "YOU WIN £";BT
 220 LET SB=-2*BT: LET BT=-SB: GO SUB 43
00: REM PRINT BET
 565 LET SK=10000: LET IS=50: REM STAKES
 610 LET P(1)=1: LET P(2)=1
 625 LET BT=0: GO SUB 4200: REM PRINT BE
T BOX
2740 IF A$="D" THEN  GO SUB 2900: IF CA=
0 THEN  RETURN : REM DOUBLE
2900 REM **** DOUBLE ****
2910 LET DB=1: LET SB=BT: GO SUB 4300: R
EM PRINT BET
2915 IF CA=1 THEN  LET DB=0: RETURN
2920 LET FL=0: LET PL=1: GO SUB 1300: RE
M DEAL
2930 GO SUB 800: REM EVALUATE
2940 LET DB=0: RETURN
4000 REM **** TITLE ETC ****
4010 CLS
4020 LET TX=10: LET TY=3: GO SUB 900: PR
INT " ZX SPECTRUM"
4030 PRINT TAB TX+3;"PONTOON"
4040 PRINT TAB (TX+5);"BY"
4050 PRINT TAB TX-7;"PETE SHAW & STEVE C
OLWILL"
4060 LET TX=5: LET TY=8: GO SUB 900: PRI
NT "YOUR STAKE IS £";SK
4070 LET TX=5: LET TY=12: GO SUB 900: PR
INT  FLASH 1;"PRESS ANY KEY TO PLAY"
4080 LET A$=INKEY$: IF A$="" THEN  GO TO
 4080
4090 RETURN
4200 REM **** PRINT STAKE ****
4210 PRINT AT 18,15;"YOUR BET"
4220 PRINT AT 20,15;"REMAINING STAKE"
4230 RETURN
4300 REM **** PRINT STAKE ****
4302 LET CA=0: REM CAN'T AFFORD TO DOUBL
E FLAG
4305 IF SK>=50 OR BG=0 THEN  GO TO 4310
4306 LET A$="": GO SUB 700: INPUT "OUT O
F CASH!  PLAY AGAIN (Y/N)";A$
4307 IF A$="Y" THEN  RUN
4308 STOP
4310 LET SK=SK-SB: LET BT=BT+SB
4315 IF DB=1 AND SK<0 THEN  GO SUB 700:
PRINT  FLASH 1;"YOU CAN'T AFFORD IT!"
4317 IF DB=1 AND SK<0 THEN  LET BT=BT-SB
: LET SK=SK+SB: LET CA=1: RETURN
4320 IF SK<0 THEN  LET BT=SK+SB: LET SK=
0: GO SUB 700: PRINT  FLASH 1;"YOU CAN O
NLY AFFORD £";BT
4340 LET TX=24: LET TY=18: GO SUB 900: P
RINT S$( TO 15)
4345 LET TX=24: LET TY=18: GO SUB 900: P
RINT "£";BT
4350 LET TX=24: LET TY=20: GO SUB 900: P
RINT S$( TO 15)
4355 LET TX=24: LET TY=20: GO SUB 900: P
RINT "£";SK
4360 RETURN
```

Commodore 64

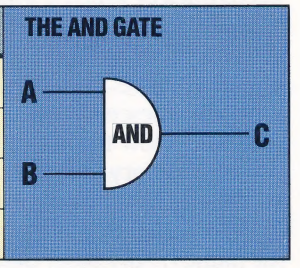
```
  72 BG=1:SB=IS:GOSUB4300:BG=0:REM PRINT B
ET
  73 AN$="":GOSUB700:PRINT"BUY A CARD (Y/N
) ";
  74 GET AN$:IF AN$="" THEN 74
  75 IF AN$<>CHR$(13)THEN PRINT AN$
  77 IF AN$<>"Y" THEN 85
  78 AN$="":GOSUB700:INPUT"YOUR BET (MAX £
1000)";AN$
  79 IF VAL(AN$)>1000 THEN 78
  80 SB=VAL(AN$):GOSUB4300:REM PRINT BET
 210 GOSUB700:PRINT CHR$(156);"YOU WIN £"
;CHR$(5);BT
 220 SB=-2*BT:BT=-SB:GOSUB4300:REM PRINT
BET
 565 SK=10000 :IS=50:REM STAKES
 610 HP(1)=1:HP(2)=1
 625 BT=0:GOSUB4200:REM PRINT BET BOX
2740 IF AN$="D" THEN GOSUB 2900:IF CA=0
THEN RETURN
2900 REM **** DOUBLE ****
2910 DB=1:SB=BT:GOSUB4300:REM PRINT BET
2915 IF CA=1 THEN DB=0:RETURN
2920 FL=0:PL=1:GOSUB1300:REM DEAL
2930 GOSUB800:REM EVALUATE
2940 DB=0:RETURN
4000 REM **** TITLE ETC ****
4010 PRINT CHR$(147):REM CLEAR SCREEN
4020 TX=13:TY=3:GOSUB900:PRINTCHR$(156);
"COMMODORE 64"
4030 PRINTTAB(TX+3);"PONTOON"
4040 PRINTTAB(TX+5);"BY"
4050 PRINTTAB(TX);"STEVE COLWILL"
4060 TX=9:TY=8:GOSUB900:PRINTCHR$(28);"Y
OUR STAKE IS £";SK
4070 TX=10:TY=12:GOSUB900:PRINTCHR$(5);"
PRESS A KEY TO PLAY"
4080 GET A$:IF A$=""THEN 4080
4090 RETURN
4200 REM **** BETTING BOX ****
4210 TX=24:TY=18:GOSUB900:PRINTCHR$(156)
;"YOUR BET"
4220 TX=24:TY=20:GOSUB900:PRINT"REMAININ
G STAKE"
4230 RETURN
4300 REM **** PRINT STAKE ****
4302 CA=0:REM CAN'T AFFORD TO DOUBLE FLA
G
4305 IF SK>=50 OR BG=0 THEN 4310
4306 AN$="":GOSUB700:PRINTCHR$(28);:INPU
T"OUT OF CASH! PLAY AGAIN (Y/N)";AN$
4307 IF AN$="Y" THEN RUN
4308 END
4310 SK=SK-SB:BT=BT+SB
4315 IFDB=1ANDSK<0THEN GOSUB700:PRINTCHR
$(28);"YOU CAN'T AFFORD IT!"
4317 IFDB=1ANDSK<0THENBT=BT-SB:SK=SK+SB:
CA=1:RETURN
4320 IF SK<0THENBT=SK+SB:SK=0:GOSUB700:P
RINTCHR$(28);"YOU CAN ONLY AFFORD £";BT
4340 TX=24:TY=19:GOSUB900:PRINT LEFT$(SP
$,15)
4345 TX=24:TY=19:GOSUB900:PRINTCHR$(5);"
£";BT
4350 TX=24:TY=21:GOSUB900:PRINT LEFT$(SP
$,15)
4355 TX=24:TY=21:GOSUB900:PRINTCHR$(5);"
£";SK
4360 RETURN
```

TRUTH TABLE

A *truth table* shows all the possible states of the inputs to a combinational logic circuit, together with their results, after a particular Boolean operation has been performed. These results will be shown as being either true or false.

The simplest form of truth table is one containing a pair of inputs which can be either on or off. When they are combined together, the output result is placed in the table thus:

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



The truth table shown describes an AND gate. A true result (1) is output along line C when both input A and B are true. If either of the inputs is false (0), then the output will also be false.

It is, however, possible to produce truth tables for more complex logic circuits. Indeed, it is here that truth tables are more often used, as it is necessary to ensure that all combinations of the inputs that occur produce the correct output for the application for which the circuit has been designed.

TTL

An abbreviation for transistor-transistor logic, *TTL* refers to a type of circuit that operates through bipolar transistors, used most commonly in mini and mainframe computers and in microprocessors. Such circuits are used in computers for their high speed and reliability. TTL logic uses +5v and 0v to represent logical 1 and 0 respectively.

TURING MACHINE

In reality the *Turing Machine* is not a machine at all but a theoretical model developed by the mathematician Alan Turing in the 1930s. The purpose of the Turing Machine is to discover whether a problem can be solved by a computer or not — whether, in present-day parlance, it is 'Turing computable' or not.

The Turing Machine may be thought of as an infinite length of tape on either side of a read/write head. This tape holds the data as discrete symbols along the length of the tape, grouped in units of five, known as 'quintuples'. By assembling a number of quintuples, an algorithm program can be constructed.

The first symbol in the quintuple indicates which 'state' the machine is currently in, while the second should match the symbol under the read head of the data tape. If the symbols do not correspond then the machine will 'fall through' to the next quintuple. The third position of the quintuple indicates the symbol that should be written to the data tape. If the symbols do not correspond then the machine will 'fall through' to the next quintuple. The third position of the quintuple indicates the symbol that should be written to the data tape. The fourth position indicates the state to which the machine should move if the tape symbol and that in the second position produces a match. Finally, the last position tells the machine whether the tape should move to the right, the left or remain where it is.

Any 'Turing computable' problem (one that can be solved by a computer) can be broken down into these steps. Note that each of the quintuples forms a simple binary problem: 'If there is a match then perform an action'. If the problem cannot be reduced to these simple algorithms then it is unsolvable by a computer.

TURNAROUND TIME

This is the time required between the commencement of a job to be carried out on a computer system and its completion, including the output of any files. The *turnaround time* can also refer to the time it takes to reverse the transmission of a data signal.

TWO'S COMPLEMENT

Two's complement is a system devised to overcome the problem of addition and subtraction of signed binary numbers. To indicate that a binary number is negative it is conventional to set bit seven (the most significant bit or MSB) to 1. The problem with this convention is illustrated by the following calculation:

```
    00011000  24
+   10000110  -6
=   10011110 -30
```

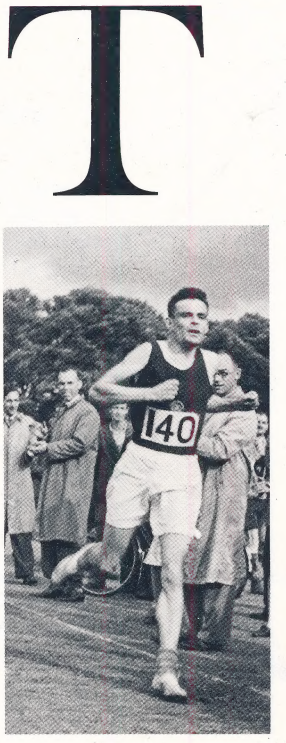
To include the sign bit within the calculation yields an incorrect result. An improvement in the calculation is achieved by using the one's complement (see page 1168) method of addition. Here we reverse all the digits in the second number but retain the value of the MSB. Thus the binary version of -6 becomes 11111001. When we perform the addition now:

```
    00011000  24
+   11111001  -6   (one's complement)
= (1)00010001  17
```

the highest bit 'falls off' the end of the byte and we are left with a result of 17 — still incorrect, but closer to the real answer. Using the one's complement system we will always obtain a result one below the correct answer. The solution, then, is to perform a one's complement addition but with one added — hence the name 'two's complement'. The calculation using the two's complement system is thus:

```
    00011000  24
+   11111010  -6   (two's complement)
= (1)00010010  18
```

which is the correct answer.



Front Runner
Alan Turing (1912-1954) was responsible for much of the theoretical work that laid the foundations for the development of the microcomputer. He was a solitary man who found inspiration and relaxation in long-distance running. In 1952, he was convicted on charges relating to his homosexuality, and he committed suicide two years later by eating an apple he had injected with poison. The Enigma of Intelligence (ISBN 004-5100-608, £5.95), a biography of Turing by Andrew Hodges, has been one of the biggest selling computer-related books of recent years

PARENTAL INFLUENCE

On a multi-user operating system such as Unix, facilities for handling files and directories must be efficiently designed and implemented. The tree structure, on which the Unix directory is based, is examined here, along with several of the more important commands.

In the previous instalment (see page 1784) we saw how every user on a multi-user operating system, such as Unix, has their own special area of disk storage, known as a *directory*, in which they can keep their own programs and data protected from interference from other users by a password system. Additionally, it's also necessary to have the following: 'public directories' containing system programs available to everyone; 'system directories', which keep, among other things, information on users and their directories; and the ability to present more than one directory so that users can keep different aspects of their work separate.

This means that there could be a very large number of directories with users having varying degrees of access to them. For example, all users will want to move freely about among their own directories while being able to run programs on public directories, but only some will have the freedom to make changes to these programs.

It's a good idea to know what other directories there are, and even what files are on them. It must also be possible to move or copy files between directories, although not necessarily to be able to make any changes to them. Unix has a rather complicated directory structure, enabling users to create new directories and delete those that they have created. It also includes commands for moving around to different directories and transferring files.

The directory structure used by Unix is the *tree* structure, used throughout computer science for organising sets of data with complex interrelationships. Each directory can contain a number of files and a number of sub-directories, all of which are given names so that they can be referred to. Although in many situations Unix will treat sub-directories and files alike, it nevertheless keeps track of which is which. Such a sub-directory is a *descendant*, or *child*, directory of the *parent* directory that holds its name. Each directory will therefore have one parent and an unspecified number of descendants.

If we follow the chain upwards looking at the parent directory each time, we must come to a halt eventually at a directory with no parent. This directory is known as the *root* and is referred to by the name /. Any other directory can be referred to by its *pathname*, which is a list of directories starting at the root, separating each directory name with /.

The diagram shows a simplified Unix directory system. The full name for the directory of user John is /usr/dept1/John. A file also has a full name, which consists of the pathname to the directory that contains it, followed by /filename. It's not normally necessary to use a full pathname to specify a particular file — not only can it often be abbreviated, but it can be omitted entirely when working solely within the user's own directory.

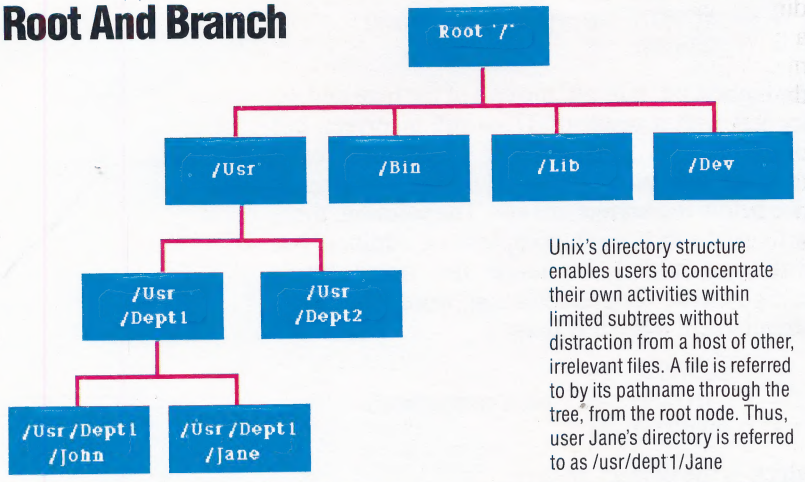
If you need to find out the full pathname of the directory you are in, the command pwd (print working directory) will provide it for you.

FREEDOM OF CHOICE

Unix has more freedom than nearly every other operating system in the names that can be used for files and directories. Up to 14 characters can be used, and it is possible to use any characters including spaces — but it is advisable to avoid certain special characters that Unix reserves for particular purposes. These are:

\ / " ' * ; - ? [ ] ( ) ~ ! $ { } < > :

Root And Branch



Unix's directory structure enables users to concentrate their own activities within limited subtrees without distraction from a host of other, irrelevant files. A file is referred to by its pathname through the tree, from the root node. Thus, user Jane's directory is referred to as /usr/dept1/Jane

CAROLINE CLAYTON

List Options

The following can be used in conjunction with the ls command:

-a	Lists all entries, including system files
-c	Lists by time of file creation
-l	Full listing
-m	Stream output separated by commas
-r	Reverse order
-s	Giving size in blocks
-F	Marks directories with a / and executable programs with *
-R	Also lists contents of all sub-directories

Options can be combined (-a and -r, for example, can be joined to make -ar)

Dialogue With Unix

```
Berkeley 4.2 Vax/Unix (infsc3)
Type <Ctrl-D> to disconnect

login:com-mcc
Password:
(note password is not echoed on screen)

You are a Normal user  (Class 3)
Jobs : 19  Superiors : 2  Maximum : 21
Last login: Fri Oct 18 11:45:37 on ttyn05

          Welcome to the Information Sciences VAX/UNIX System.

%pwd
/mnt/com/com-mcc          (this is my home directory)
%cd fred                  (moving to a subdirectory)
%ls                       (list all files in this directory)
rec.c   receive   rx.p   transmit   tx.p
%ls -a
.   ..   rec.c   receive   rx.p   transmit   tx.p

(note the presence of two extra names '.' which is this directory and '..'
which is the immediate parent of it)

%ls -l
total 42
-rw-rw-r--  1 com-mcc      502   Sep 17 12:07 rec.c
-rwxr-xr-x  1 com-mcc    18432   Oct 21 11:02 receive
-rw-r--r--  1 com-mcc     1068   Oct 18 14:44 rx.p
-rwxr-xr-x  1 com-mcc    19456   Oct 21 11:03 transmit
-rw-r--r--  1 com-mcc     1244   Oct 21 11:01 tx.p

(the three groups of 'rwx' indicate access rights for 1.the owner of this
directory, 2.others in the owner's group and 3.all other people.
r means reading allowed, w means writing allowed and x means execution
allowed. A dash means that access of that type is not allowed.)

%ls ?x.p                  (using wild card characters)
rx.p   tx.p
%ls r*
rec.c   receive   rx.p
%ls *.[pc]
rec.c   rx.p   tx.p

%mkdir mike               (making a new subdirectory)
%ls -l
total 43
drwxr-xr-x  2 com-mcc      24   Oct 21 11:10 mike
-rw-rw-r--  1 com-mcc     502   Sep 17 12:07 rec.c
-rwxr-xr-x  1 com-mcc   19456   Oct 21 11:03 transmit
-rw-r--r--  1 com-mcc    1244   Oct 21 11:01 tx.p

(note that a new subdirectory is indicated by a 'd'.)

%cd mike                  (change working directory to new one)
%pwd
/mnt/com/com-mcc/fred/mike
%cd ..                    (back to parent directory)
-rwxr-xr-x  1 com-mcc    18432   Oct 21 11:02 receive
-rw-r--r--  1 com-mcc     1068   Oct 18 14:44 rx.p
%pwd
/mnt/com/com-mcc/fred
%cp rx.p mike             (copy file 'rx.p' to the new directory)
%mv tx.p mike             (move file 'tx.p' to new directory)
%ls                       ('tx.p' has now gone from this directory)
mike   rec.c   receive   rx.p   transmit
%cd mike
%ls
rx.p   tx.p
%who                      (find out who else is using the system)
root              console Oct 21 08:07
com-rgd           ttyn00  Oct 21 09:45
ccs-klf           ttyn01  Oct 21 09:45
cs4bc             ttyn04  Oct 21 10:57
com-mcc           ttyn05  Oct 21 10:58
ccs-mdb           ttyn06  Oct 21 10:06
cs4cy             ttyn07  Oct 21 11:06
com-ah            ttyn08  Oct 21 11:00
com-jh1           ttyn10  Oct 21 11:05
cs4bg             ttyn11  Oct 21 11:05
%finger com-mcc           (details on another user)
Login name : com-mcc    Real name : curtis
Department : Not known
Directory : /mt/com/com-mcc
Shell : /bin/csh
Logged in at 10:58 on Mon Oct 21 on ttyn05   36 secs idle   No Plan.
%logout                   (finish this session)
Host sent disconnect
```

When referring to file and directory names, Unix has a wild card system for abbreviating sets of names. The following wild card characters can be used:

- ? is used to match any single characters.
- * will match any group of characters, excluding a full stop as the first character in a name. This is to protect system files such as .login or .cshrc from accidental erasure.
- [] (with a list or range of characters inside the brackets) will match any single character with one of the characters inside the brackets; such as [a,e,f], [A-Z] and so on.

To see how these work, let's consider some of the commands available for file manipulation.

The ls command is used to list the files and sub-directories within a directory, and like many Unix commands, ls can take a number of options (the complete set is shown in the box). There are two commands that allow you to list the contents of a file; cat, which can also be used to concatenate files, and more, which allows you to view only a screenful of information at a time.

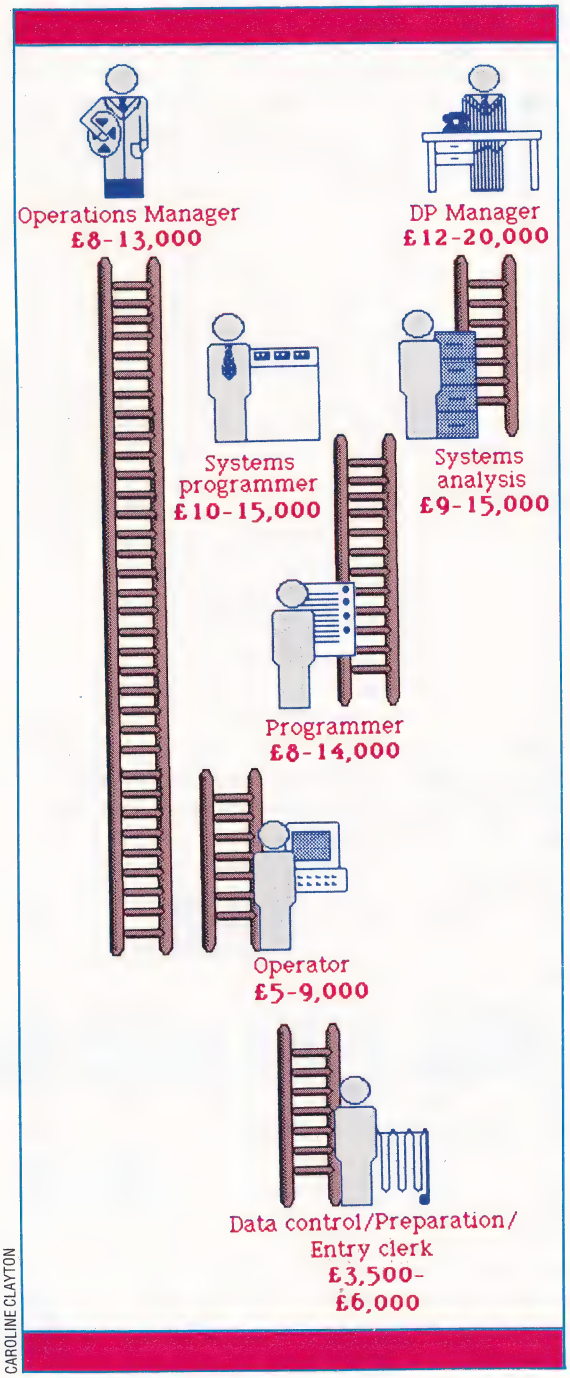
Users can create and remove their own sub-directories at will. The command mkdir will create a new sub-directory of the current directory, and rmdir will delete it, provided it has no files or sub-directories left in it.

The current working directory can be changed using the cd command. If the new working directory is a sub-directory of the old one, the directory name is given; but if the new directory is somewhere else, a full pathname must be given. The use of cd without a directory name always takes you back to the home directory (the one to which you originally logged in).

We give an example Unix session using some of these commands. Comments have been inserted (enclosed in braces { }) to explain what is happening — they do not occur in the actual dialogue.

DP OPPORTUNITIES

Over the last 30 years, data processing has grown into a large, powerful and well-paid industry, with a seemingly never ending demand for new workers. As a consequence, this expanding sector of the job market offers a wide variety of career opportunities, many of which we look at in this instalment.



CAROLINE CLAYTON

Ladder To Success
DP offers a number of differing job opportunities, ranging from the managerial role of the department manager to the intellectual challenge of systems programming. Our diagram shows the different career paths available, together with an indication of the salary range for each job

Each week, *Computing* and *Computer Weekly* carry over 100 pages of job advertisements offering careers in data processing with salaries ranging from £10,000 to £15,000 and upwards. Let's look at the opportunities this field really offers, what the career paths are and how the novice can enter the industry.

We should first ask what data processing actually consists of. Essentially, DP involves the manipulation, retrieval and storage of information relevant to an application. Much of DP revolves around routine records — dealing with a company's accounts or payroll, for example.

A career in DP doesn't mean you will automatically get your hands onto the latest piece of microcomputing hardware. On most DP sites, the micro is still seen as a toy and 90 per cent of the work is carried out on minicomputers or mainframes, provided by manufacturers like IBM, ICL and Burroughs.

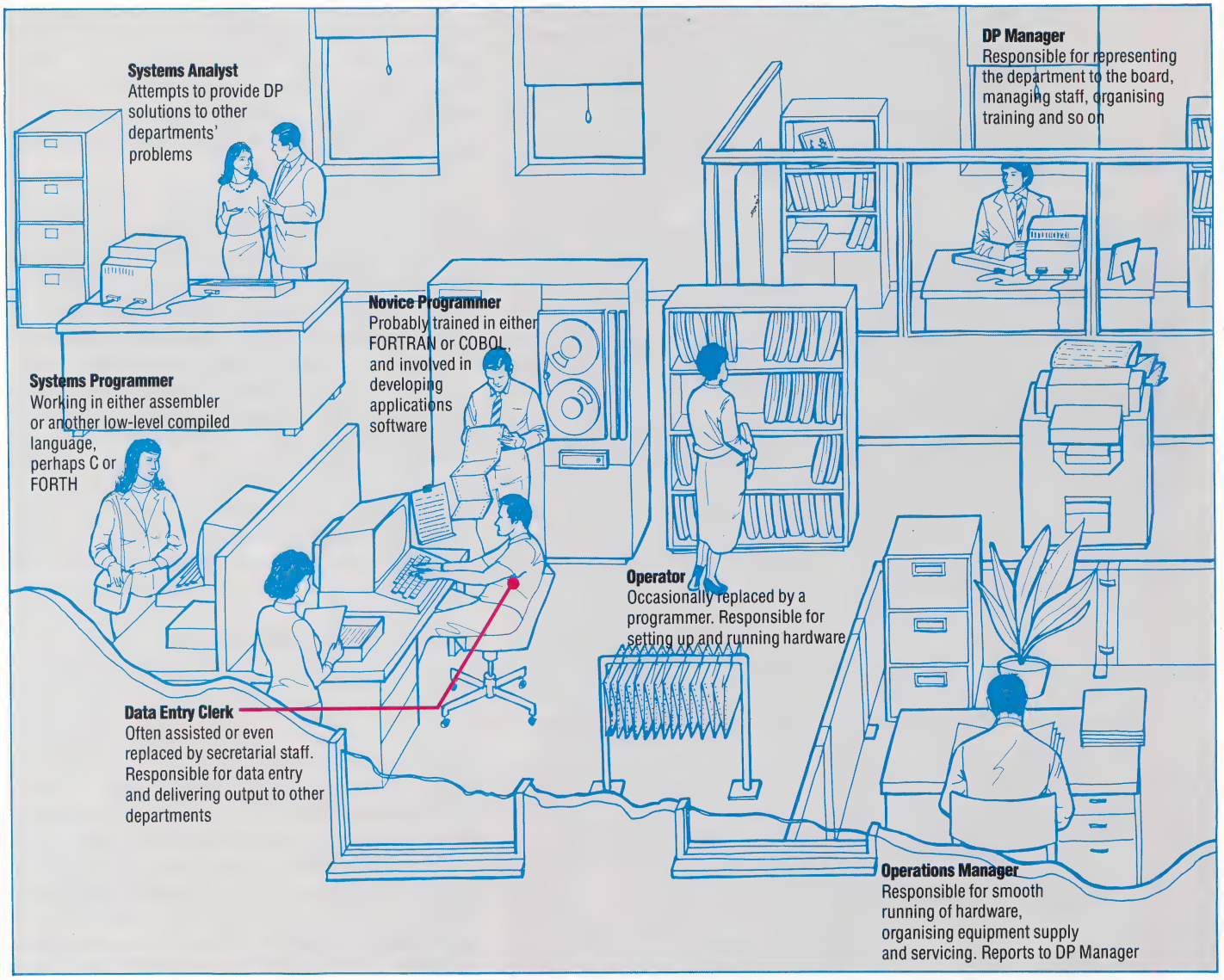
The Careers Ladder table shows the major job areas and 'rungs' in the DP sector. These are not, however, hard and fast categories. In some sites, for instance, one person will combine the role of analyst and programmer, in others that of operator and programmer. In general, however, these are the job categories into which you could sift 90 per cent of the people in the industry.

At the bottom of the ladder are the 'data preparation', 'data entry' or 'data control' clerks. These jobs involve little or no real computing knowledge and, as the salary levels show, are carried out by unskilled workers. The work effectively involves 'keying' data into a machine via the keyboard onto a disk or tape. It also involves decollating paper and dispatching it. On smaller sites, this area of work will be carried out by secretarial staff with help from operators and programmers. It's debatable whether a data preparation clerk can really be considered as one of the DP staff. On many sites, data preparation is seen as an unskilled and dead-end job that doesn't lead to a career as a programmer or operator.

THE OPERATOR'S JOB

The operator, the next step up from the data preparation clerk, is responsible for the running of the computer itself. The operator's job can therefore be likened to that of a driver of a car while the programmer can be seen as a navigator. The operator is responsible for physically initiating jobs on the machine, loading tapes and disk packs, loading paper into printers and starting machines up and switching them off.

Traditionally, it was relatively easy to transfer from the ops side to programming. But with the 'de-skilling' of the operations role and the increasing number of graduate trainee programmers, this tradition is fading fast. In fact, there is now a glut of people in operations trying to get out of the field. How easy it is to transfer from ops to programming depends very much on the attitude of the managers on the site. While being an operator for a short time shouldn't be ruled out,


Systems Analyst
Attempts to provide DP solutions to other departments' problems
DP Manager
Responsible for representing the department to the board, managing staff, organising training and so on
Novice Programmer
Probably trained in either FORTRAN or COBOL, and involved in developing applications software
Systems Programmer
Working in either assembler or another low-level compiled language, perhaps C or FORTH
Operator
Occasionally replaced by a programmer. Responsible for setting up and running hardware
Data Entry Clerk
Often assisted or even replaced by secretarial staff. Responsible for data entry and delivering output to other departments
Operations Manager
Responsible for smooth running of hardware, organising equipment supply and servicing. Reports to DP Manager

KEVIN JONES

Action Stations
Our illustration shows the interior of a typical DP department in a large company. The position of such departments has been significantly eroded over the years as more departmental heads get access to their own desk-top micros. Nevertheless, DP remains a major source of employment and offers significant career opportunities

it is important to ascertain attitudes to promotion before taking it on.

Most people looking to have a long-term career in DP will try to step on the ladder at the next highest rung — the programmer. Compared to programming on a micro, however, programming in a DP environment tends to be a much more specialised activity.

As a novice programmer, you will probably learn either COBOL or FORTRAN (or possibly PASCAL) enabling you to write applications software. On many sites, however, depending on the hardware used, there is another programming layer — that of systems programmer. Systems software sits between the computer's operating system and the applications software being run, and is almost always written in a machine-specific low-level language, most likely assembler. However, C is becoming quite popular in this field because of its speed and portability.

Many programmers will go on to become systems analysts, the newest of the main job roles in DP. Essentially, the systems analyst is in charge of deciding with users what kind of information they require or could benefit from. The job calls for the ability to talk to managers who may have no understanding of computing, and to explain to them what is and isn't possible. Many people distrust and fear computers, regarding them as a threat to their livelihoods. The systems analyst has to be able to chip away at this distrust and get an accurate picture of exactly what a particular job involves. For this reason, analysis calls for considerable communications skills (and a considerable amount of patience and diplomacy).

As a programmer, you have a choice of two career paths — become a software specialist or a systems analyst. Which route you choose to follow depends very much on your character and outlook on life. If you are a loner who doesn't really need a lot of human contact in your working life, then you may be happy to stay in programming. Indeed, programming, if you find the right area to specialise in, can be extremely lucrative and very intellectually rewarding. If, on the other hand, you like working with others and the idea of staring at a screen for the rest of your life sounds worse than dismal, systems analysis may be the answer.

The DP manager carries out three functions. With the aid of systems analysts and senior

programmers, the manager chooses and evaluates new hardware and software for the site. Secondly, he is responsible for managing those working in the DP department and for ensuring that they receive adequate training. Finally, he represents the department to the outside world, reporting either to the board or to the management services manager.

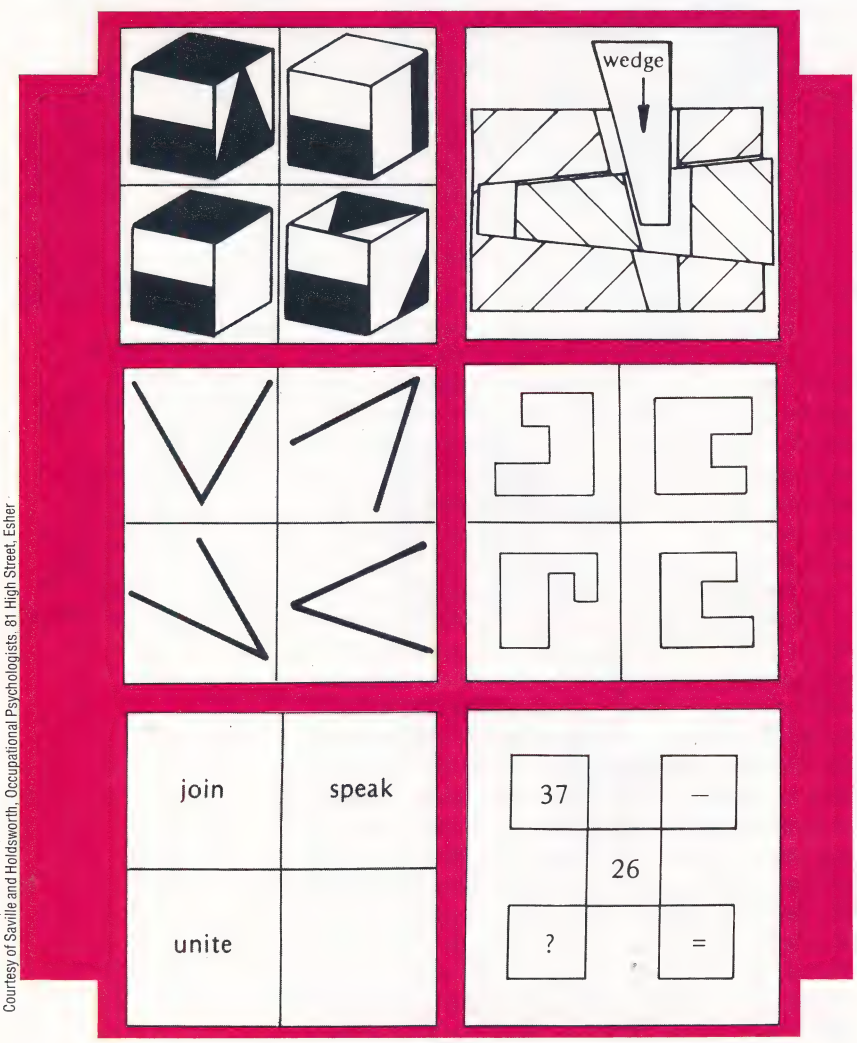
There are four main ways of becoming a programmer: you can progress from being an operator (not a route we would particularly recommend), you can enter the field from a TOPs course, as a computer science graduate, or as a graduate trainee with a degree but one which is not directly relevant (or as someone who has relevant business experience but no previous computing experience). Let's consider each of these options in turn.

The government-run TOPs courses offer a good basic training in one of the main DP languages — normally COBOL. Entry to the course is based on test results, with graduates generally being slightly favoured. But that's only the beginning. In 1982, almost everyone on a TOPs course got jobs on leaving — today that simply isn't the case, and many TOPs graduates are unemployed for months after leaving. There is also evidence of occasional discrimination against TOPs graduates — some job advertisements specifically ask TOPs people not to apply. That doesn't means that TOPs courses aren't worth doing, though. Tens of thousands have got jobs because of them. But before doing one, try to assess the success rate that the college has achieved in the past.

Unquestionably, programming (and DP in general) is rapidly becoming a graduate profession. Computer science graduates can really pick and choose, and even graduates in English or a social science should be able to get a traineeship providing they can pass the all important test. Maths or science graduates with some experience on a university mainframe are off to a flying start.

DP isn't a closed field to those who are slightly older and who have useful industry experience. Ford, and a number of other automobile companies, have trained line managers with a good knowledge of the motor industry in programming. A number of companies team experienced programmers with managers in the hope that a closer reflection of user needs will result.

Testing Testing...
Employers are often unwilling to rely solely on selection by interview and will require job candidates to complete one or more tests to determine their suitability. These tests, usually designed by occupational psychologists, are in some ways similar to IQ tests but go further in their assessment of job-related skills. Our illustration shows a stylised representation of some of the problems that might be faced by an applicant for a technical apprenticeship. From top left: spatial reasoning, mechanical comprehension, visual estimation, spatial recognition, verbal reasoning and numerical computation



Courtesy of Saville and Holdsworth, Occupational Psychologists, 81 High Street, Esher

IN-SERVICE TRAINING

Progress in a DP career depends very much on the amount of training you can get. Training in some arcane branch of systems software, for instance, could provide you with skills worth $4,000 to $7,000 a month in the USA. Even in an apparently tight economy, do try to find out what your potential employer's record is on training before committing yourself. Some companies give their staff minimum training, or expect them to learn on the job in the general belief that a well-trained employee is a footloose employee. In general, the larger the company the better the training — BA and BT have particularly good records.

Apart from training, the other main factor affecting salary levels is your machine experience. In general, it is quite hard to skip from one manufacturer's machine to another — if you have been working on ICL machines it is difficult to get a job on an IBM site. This reflects the fact that each manufacturer has developed different systems software and that these can differ considerably. It would be dangerous to get a job working on an aged machine from an unsuccessful manufacturer, since your skills would obviously be in negligible demand elsewhere. It would be best then to work on an IBM site (over 80 per cent of all mainframes worldwide are IBM). But Honeywell and Burroughs both have large user bases as well. ICL is possibly in danger of losing its grip on the UK mainframe market and would not be a good bet if you wanted to work overseas. You should think carefully before embarking upon a career involving specialisation on Univac, NAS or Kienzle hardware.

In the next instalment, we'll be looking at the retail and marketing opportunities associated with computer sales.

IN THE PIPELINE

A logical organisation of software and data can be maintained in MS-DOS by placing system utilities, applications and data files in a hierarchical tree of sub-directories. We look here at some of the ramifications of this — namely, that users must be constantly aware of both the sub-directory in which they are working and the local branches.

The Unix utilities pwd (print working directory) and tree (display the directory structure) are not always provided by OEMs supplying MS-DOS, but other alternatives are available. By including the characters $p in the prompt command, the full pathname (from the root of the current drive) will be displayed as part of the DOS prompt after every command terminates execution.

Supposing, however, that we are working in a sub-directory called, say, WP (for word processing), and we wish to format a disk and make a copy of a new document. Giving the command format b: is of no use because the transient utility FORMAT.EXE is in a sub-directory called system. We must say \system\format b:, using the full pathname. If another program, called WCOUNT.EXE (word count) was on drive C in a sub-directory several levels down, we would again have to give the complete pathname for it to be found and executed. So counting the number of words in a document called report17.doc might mean typing:

```
C:\text\tools\wcount report17.doc
```

Fortunately, there is a very easy way round this problem. The resident command path will display or create a list of default paths that will be searched by MS-DOS whenever an unrecognised command is not located in the current directory. On its own, path will display the message No path before any defaults have been specified. If the most frequently used programs and utilities were all to be found in the directories just given, we could give the command:

```
path=A:\system;C:\text\tools
```

Now, whichever disk or directory is the current one, the command processor will find any program in the current directory or either of the other two just specified. The search is conducted in the same order as the paths are listed; so, for example, if a program name was to be duplicated, the first match would be executed.

One word of warning: spaces must *not* be inserted in the argument list, since they are regarded as terminators. The command:

```
path C:\system; A:\mylib; C:\text\tools
```

would subsequently fail to find anything that wasn't either in the current directory or on drive C in the system directory.

REDIRECTION

The Unix-inspired versions of MS-DOS (versions 2.0 onwards) have the ability to 'redirect' I/O from and to both system devices and disk files. The standard system devices such as CON (the console) and PRN (the printer) are handled in exactly the same way as a text file on disk would be. Using the resident command type will normally display the contents of a text file on the screen, but the output may be redirected, with the chevron symbol (>), to any device or file we wish. For instance:

```
type report17.doc > prn
```

will initiate the printer and produce hard copy of

Further Resident Commands

We give here a further list of some of MS-DOS's resident commands (see page 1775). The following commands are all built into COMMAND.COM, the MS-DOS equivalent of CP/M's command line interpreter, or CCP

Command	Function
cls	Clear the screen
prompt	Change the system prompt
pwd	Print working directory
re (or rmdir)	Remove a directory (must be empty)
ver	Print the MS-DOS version number
vol	Print the volume ID

Not all systems will have every one of these commands, however. For instance, ped (borrowed from Unix) is not essential, since the prompt command can be used to display the current path, as in:

```
prompt $p
```

The 'p' standing for 'path' in this case. Other characters following the $ symbol in a prompt argument have special meanings, and some of these are:

d	Print the date
t	Print the time
_	Send a CR/LF to create a new line
e	Send an escape character

The escape sequence, as used by a standard ANSI terminal, may be used to change the VDU attributes, so that:

```
prompt $t on $d $_ $e[7m $p $e[m
```

will print the time and date on one line, and the current path in inverse video on the next. So:

```
15:42:36 on 17-11-85
A:\SYSTEM\UTILS
```

the file. The command:

```
dir a: > c:nov22.dir
```

would list the directory contents — not to the VDU as usual, but to a file on drive C with the name nov22.dir. By using redirection of output in this manner, any program or utility can be made to send its output to any file or device on the system.

The double chevron, >>, causes the same output redirection but without overwriting the previous contents of a file. So after:

```
dir b: >> c:nov22.dir
```

the file nov22.dir will contain a list of the files and any sub-directories in the current directories of drives A and B.

When used in isolation, this is merely convenient but, if combined with a few simple 'software tools', the use of I/O redirection can almost transform MS-DOS into an effective interactive programming language.

Some of the most effective software tools are simple programs that make some change to the data read from standard input before passing it on to standard output. These are called 'filters', from the obvious analogy with electronics. Filters can be written very easily even by a novice programmer, with the proviso that they must be written in a compiled language so the machine code produced is 'stand alone', and that they may be executed just like any system utility without the need for an interpreter or the source code to be present.

Unix and MS-DOS versions 2/3 have the ability to connect files with either of the character handling routines for the standard system input *and* output devices. To redirect input, < is used. In this case, any program that expects input data from the keyboard may be instructed to obtain it from a file. This includes both transient and resident commands. For example:

```
date < date.now
```

will accept a string representation of a date from a file called date.now — the normal output message produced by the command appearing on the screen as usual. This can be very useful if only a software clock is available, which is reset every time the system is rebooted. Including the previous line in the autoexec.bat file saves having to retype the date continually.

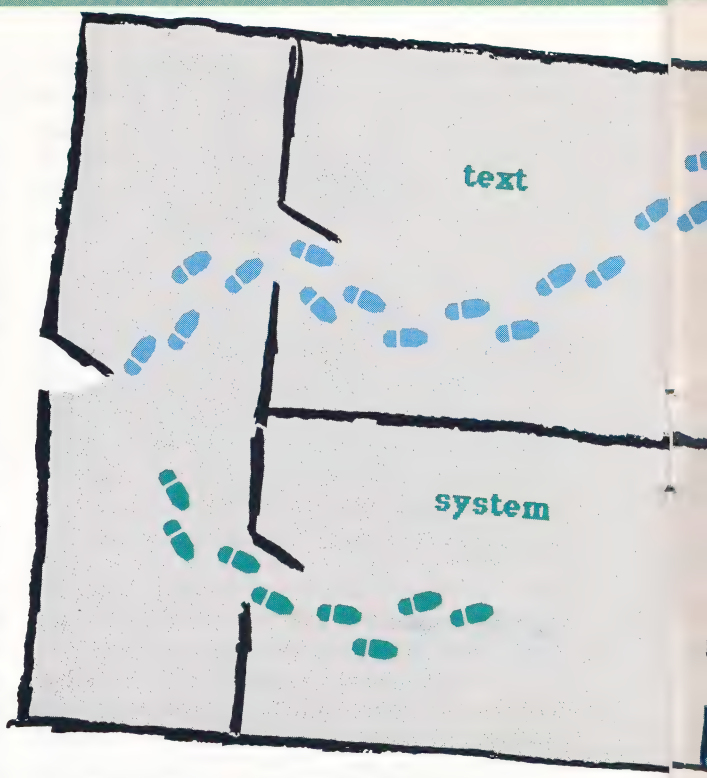
There are some useful filters supplied as MS-DOS transient utilities. One of them (MORE.COM) takes any input text and simply copies it to the standard output. After every 23 lines, however, MORE displays the message:

```
--More--
```

and waits for a key to be pressed, thus enabling a screenful of text at a time to be viewed at leisure — for instance:

```
more < report.doc
```

Naturally, output redirection is not especially useful with MORE.



One of the most useful MS-DOS transient utilities is SORT.EXE. This is a filter that reads lines of text and, before copying them unaltered to the standard output, sorts them into alphabetical order. If we wanted a sorted directory listing of all text files, for example, this could be achieved with:

```
dir *.txt > temp.dir
sort < temp.dir
del temp.dir
```

MS-DOS, like Unix, allows these processes to be performed in one operation by means of a 'pipe" or 'pipeline'. The | symbol is used to indicate that the output of one process should be redirected to a

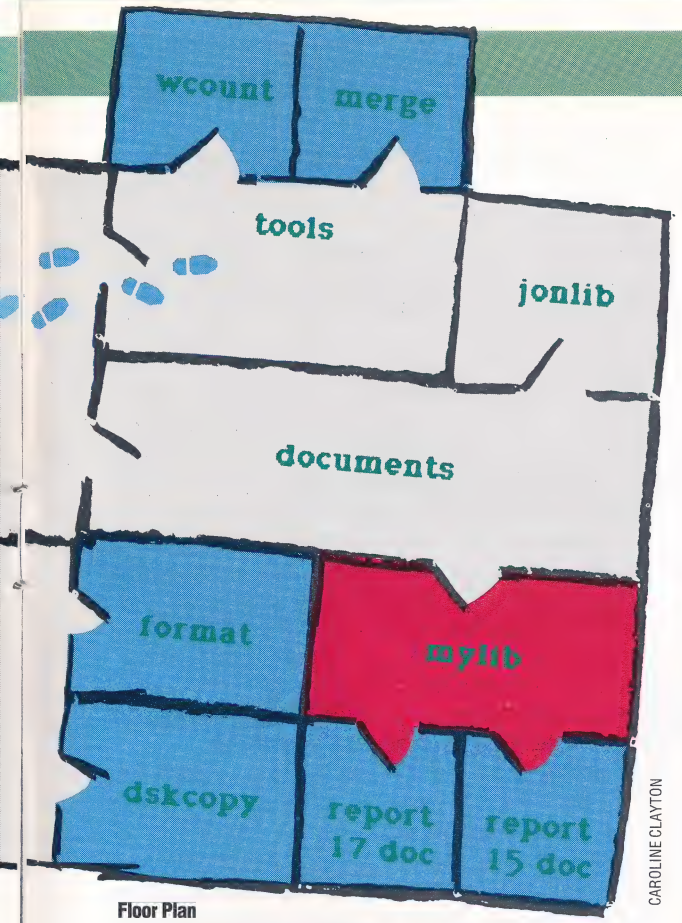
Filtration Plant

A simple 'filter' can be a program that merely takes standard input and alters, adds or removes certain characters before passing the (filtered) result on to the standard output. Consider a simple filter (called pipz) to prevent characters with ASCII codes of 128 and upwards getting to the output.

Text files produced with WordStar and certain other programs will often contain such characters, since the most significant bit can be set for formatting purposes. A 'cleaned' copy of a test file that had originally contained such codes can easily be made with a command to redirect both the input and output channels:

```
pipz < badfile.txt >goodfile.txt
```

This simple program (in a PASCAL version) 'folds' all character values to the normal ASCII range of codes (up to 127) and thus emulates the use of the [z] option with CP/M's pip utility, zeroing the high bit of each character byte. This is particularly useful because the MS-DOS copy command does not have



CAROLINE CLAYTON

Floor Plan
The hierarchical file structure of MS-DOS can be thought of as a series of interconnecting rooms. Entry to certain rooms can only be made through others, reflecting the tree structure of the directory system. 'End' rooms represent actual files or programs, rather than directories, and are coloured blue on the diagram. Let's say that we were working within the sub-directory called 'mylib' (coloured red) on a text file called 'report17.doc'. Utility programs such as word counters (wcount) and disk formatters reside in other 'rooms'. To avoid having to specify the complete path from the root each time we wished to use a utility, MS-DOS provides a path command which defines default paths. To allow us direct access to the utility directories we could issue the command: path \system;\text\tools, which would specify two alternative search paths for the utility we required

channel or stream that is to act as the input source for a subsequent process. In the case just illustrated, therefore, we need only say:

```
dir *.txt | sort
```

Any required temporary files are handled invisibly by MS-DOS, and are directed automatically when a pipeline terminates.

The command sort may take options including /r (to sort in reverse order) and /+n (where n is any number), indicating the character position on each line at which comparisons for sorting should start. The MS-DOS directory details include the file sizes in bytes (starting at the 15th character position) and also the time and date of creation. We could therefore produce a hard copy listing of the directory sorted with the largest files listed first:

```
dir | sort /r /+15 > prn
```

Another powerful DOS filter (FIND.EXE) can be used, for example, to find strings in any given text files, which can be performed simply with:

```
find "sort " C:\articles\MSDOS4.txt
```

Each line of the file containing the string "sort " is listed to the output (once only, even for multiple occurrences on the same line). Notice that the space following "sort means that sorted and sorting are not found. Nor, because find is case-sensitive, would Sort be located.

This deficiency can easily be overcome by writing our own simple filter (LOWER.EXE) to convert all input to lower case. Piping it 'in series' before other piped commands instantly cures all the other tools of their case sensitivity:

```
lower < meeting.doc | find "metal " | more
```

would list every line in the document (meeting.doc) that made reference to metal, metallic or metallurgy (whether capitalised or not) and pause after every 23 such lines had been displayed.

The power of pipes and redirection is well illustrated with commands such as:

```
dir *.pas | find "-08-85" | sort > ptn
```

which prints all PASCAL source (.pas) files created in August in alphabetical order. The output of find may be listed with the line numbers prefixed (the /n option) or the display of lines can be suppressed and just a total count of the number of occurrences obtained with /c. The /v option finds all lines *not* containing the string.

Suppose we have a text file (company.dat) containing some names and other details, including salaries, starting in column 25:

```
lower < company.dat | find "smith"/v | sort /+25 | more
```

would produce a list of everyone *not* called Smith or Smithers and so on, sorted in order of decreasing salary.

By using pipes and redirection, we have effectively created a new program on the command line by building it on the spot from simple filtering software tools.

this option. The file could equally well be listed direct to the printer with the command:

```
pipz < badfile.txt >prn
```

An example filter in PASCAL:

```
PROGRAM PipZ (input,output);
{fold chars to ASCII range 0..127}
VAR
   c :char;
BEGIN
   WHILE NOT EoF (input) DO
      BEGIN
         WHILE NOT EoLn (input) DO
            BEGIN
               read (input, c);
               write (output,
                  chr (ord (c) MOD 128))
            END;
         ReadLn (input);
         WriteLn (output)
      END
END.
```

POINTING THE WAY

The pointer, a special variable type provided by C, may be used to reference absolute memory addresses. We examine here the declaration procedure and the use of this powerful facility before taking a closer look at string handling.

Designed as a replacement for assembly language, C has a range of features that other high-level languages do not possess. One of these is the facility to refer directly to machine addresses. PASCAL has 'pointer' types, but the correspondence between these and machine addresses is not at all clear. In C, the pointer type does refer to actual addresses, though it must be remembered that on a multi-tasking machine, particularly one that uses virtual memory, actual knowledge of the numerical address of a variable is not normally particularly useful.

Nevertheless, C has the capability to refer to absolute addresses and so can be used for driving hardware devices directly. Every variable, of whatever type, has an address that can be assigned to a pointer. It is often more convenient, particularly with strings (arrays of characters), to use pointer arithmetic to access elements of the string rather than to use the normal array subscripting.

The * character is used to declare a pointer variable, so:

```
int *p ;
```

would declare a pointer variable p. Note the int declaration — pointers can only point to a particular variable type (the 'base' type), so if we wanted a pointer to access variables of type char, we would need to enter:

```
char *p ;
```

and so on. The * character, when used in this fashion, simply means 'create a pointer', so at this stage the pointer itself does not actually point anywhere at all.

To initialise a pointer to the required value, we use the ampersand character — & — which, when placed in front of a variable, has the effect of returning the address of the variable rather than its value. So:

```
p = &i ;
```

would set up p as a pointer to i. At this point, you should note that the * character can also be used to indicate the value pointed to by a pointer variable. In this case:

```
i = *p ;
```

would store the *value* pointed to by p in i, whereas:

```
i = p ;
```

would store the *address* pointed to by p. You will, of course, need a 'cast' to avoid compiler error messages when handling absolute addresses, as in:

```
p = (int*) 1000;
```

The increment and decrement operators ++ and -- can be used with pointer variables, and they increment or decrement by the appropriate amount for the type of value being pointed at. On a system that uses four-byte integers, therefore, if p is a pointer to the first element of an array of integers, then p++ will increment the address in p by four to point to the next element. The same thing would happen if we used p=p+1: it would add on the size of the appropriate data item rather than just one. This emphasises the fact that pointers are not the same thing as integers, even if the values stored therein are actually integer numbers.

Pointers can be passed as parameters in function calls. This gives the facility to pass by reference — that is, the value of the pointer will be passed to a local variable in the function, but this value will still point to the same item as it did in the calling program. Thus, changes made to the stored value will still be there when control is returned.

There is a very close correspondence between pointers and arrays. When an array is declared, the unsubscripted array name is, in fact, a pointer to the first element in the array. In other words, given int a[100], *p, then p = a; and p = &a[0]; are entirely equivalent. This means that access to elements in the array (or to sub-arrays) can often be done more efficiently by arithmetic on pointers rather than by subscripts.

USING STRINGS

Strings are simply one-dimensional arrays of char. Since their use is fundamental to many applications, however, C provides a few special facilities and library functions to carry out the normal types of string processing. In C, a string consists of an array of char in which the actual characters that make up the string are followed immediately by the null character \0 (ASCII zero). This may occur at any point in the array, so we have at least the illusion of dynamic strings of varying lengths even if the underlying arrays must be of fixed length. Remember that the \0 character will take up one character position in the array, so it must be defined with at least one more element than the maximum number of characters.

String constants may be used, if they are enclosed in double quotes, and may be assigned to an array of char of adequate length — the \0 terminator will be added automatically. The variable to which they are assigned can be either an array or a pointer to char:

```
char *s;
*s = "abc";
```

causes the four characters a, b, c and \0 to be stored

String Handling Functions

strcmp(s1,s2)—s1 and s2 are pointers to char (strings), an integer value is returned which is less than zero if s1<s2, zero if s1=s2 and greater than zero if s1>s2
strncmp(s1,s2,n)—Similar to strcmp except that, at most, n characters are compared
strlen(s)—Where s is a pointer to char, returns the (integer) length of the string
index(s,c)—Where s is a pointer to char and c is a char, returns a pointer to the first occurrence of c in s, or NULL if it does not occur
rindex(s,c)—Similar to index except that the search is carried out from the rightmost character
strcat(s1,s2)—Appends a copy of s2 at the end of s1, returning s1. It is assumed that s1 is long enough to contain all the characters
strncat(s1,s2)—Appends at most n non-null characters to the end of s1, returning s1
strcpy(s1,s2)—Copies the contents of s2, up to the '\', into s1. It is assumed that s1 is long enough. Any previous contents of s1 are lost. The value of s1 is returned
strncpy(s1,s2,n)—Copies at most n characters from s2 into s1, appending a '\' unless n >= the length of s2

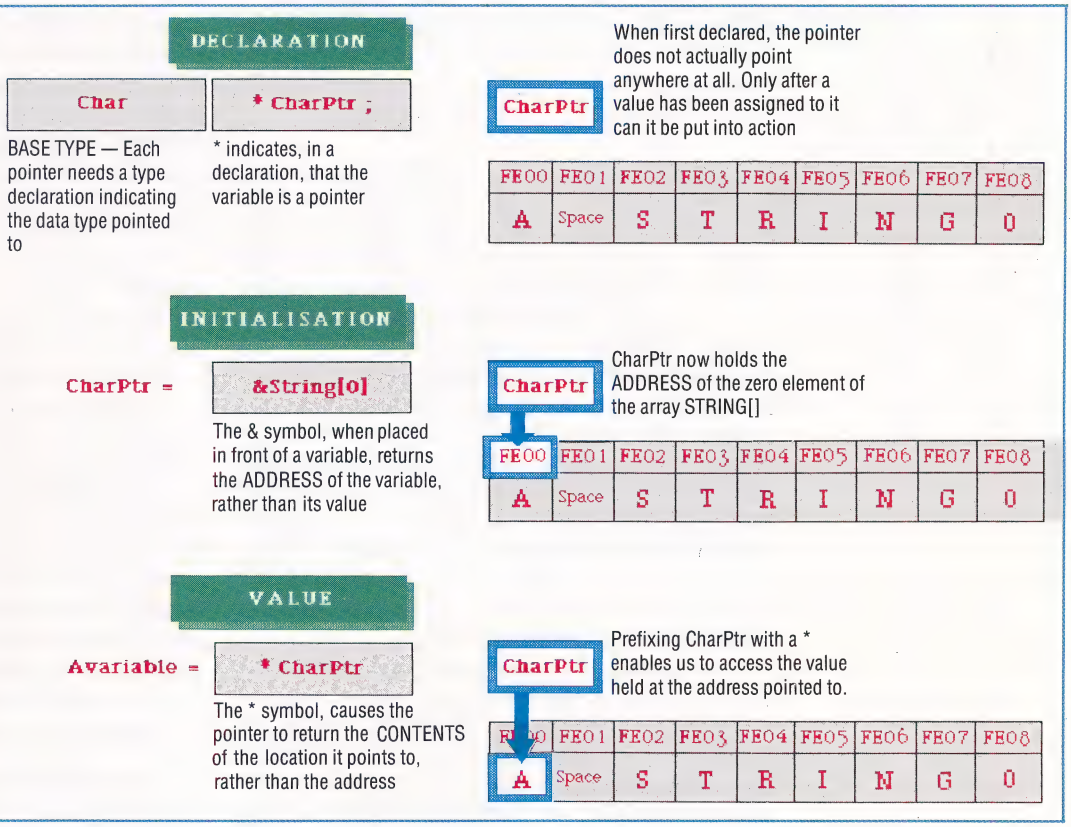
Bubbling C

```
bubble(a,n)
int a[],n,putinorder();
/* a is array of n integers to be sorted */
{
int i,j;
   for (i = o; i < n - 1; ++i)
      for (j = n - 1; i < j; --j)
         putinorder(&a[j - 1],&a[j]);
/* passing addresses of the array elements as
parameters */
}
putinorder(x,y)
int *x,*y,swap();
/* x and y are pointers to integers */
{
   if (*x > *y)
      swap(x,y);
}
swap(x,y)
int *x,*y;
{
   int temp;
   temp = *p;
   *p = *q;
   *q = temp;
/* note that the addresses of the two integers are
passed by value but the function can refer to the
actual values stored there even though they are
strictly outside the scope */
}
```

Pointing By Numbers
Our listing shows the close relationship between pointers and arrays in C, by implementing the traditional bubble sort algorithm using an array of integers. C pointers can often be more efficient than subscripts in accessing array elements

To The Point

Three stages in the use of a theoretical pointer CharPtr — used to reference different elements of the string array STRING[] — are shown here. Note that after initialisation, CharPtr returns an address, whereas *CharPtr returns the value held at that address. Pointers can be incremented and decremented to point to different elements of an array — the pointer value will be adjusted according to the 'base data type' given in the declaration statement. For example, a base type of char will result in adjustments in single-byte steps, and int would (on most systems) result in double-byte adjustments


DECLARATION
Char
* CharPtr ;
BASE TYPE — Each pointer needs a type declaration indicating the data type pointed to
* indicates, in a declaration, that the variable is a pointer
CharPtr
When first declared, the pointer does not actually point anywhere at all. Only after a value has been assigned to it can it be put into action
FE00 FE01 FE02 FE03 FE04 FE05 FE06 FE07 FE08
A Space S T R I N G 0
INITIALISATION
CharPtr =
&String[0]
The & symbol, when placed in front of a variable, returns the ADDRESS of the variable, rather than its value
CharPtr now holds the ADDRESS of the zero element of the array STRING[]
VALUE
Avariable =
* CharPtr
The * symbol, causes the pointer to return the CONTENTS of the location it points to, rather than the address
Prefixing CharPtr with a * enables us to access the value held at the address pointed to.

in four consecutive locations starting at the current address in s, which starts off pointing to the character a. After ++s, s would then point to the b, and so on.

Pointers are variables like any others so that they occupy storage space that has an address. It's therefore quite possible to have pointers to pointers and so on. This can make things rather difficult, but there are quite a few uses of arrays of pointers. One in particular is in recognising command line arguments — that is, characters that are typed on the same line as the call to the program.

There are two special parameters that can be passed to the main function, which provide access to these. They are argc, which gives a count of the number of command line arguments (arguments being separated by at least one space), and argv, which is an array of pointers to strings with argc elements. Each element in argv will point to the first character in that number string on the command line. The following short program illustrates this by simply printing out the command line arguments one per line:

```
main(argc,argv)
int argc;
char *argv[];
/* note that this declaration gives an array of pointers */
{
   int i;
   for (i = 1; i < argc; i++)
      printf("%s\n",argv[i]);
/* note that the '%s' format requires a pointer to a
string */
}
```

GETTING INTO THE SUBROUTINE

Structuring your programs to include calls to self-contained subroutines is an extremely handy way of ensuring that they are economical, adaptable and easily debugged. We discuss the 68000's JSR ('jump to subroutine') instruction, and illustrate its usefulness with a coded example.

To show the 'jump to subroutine' instruction in action, let's begin with a simple program to transform an array of 10 words length using the formula:

```
TARRAY[I] := ARRAY[I] ^2 +20*ARRAY[I]
```

and produce a sum of all the TARRAY items in a location called SUM. The formula looks a bit forbidding but really all it says is 'take each array element, square it, add on 20 times the array element, and then store the result in the corresponding element of the new array, TARRAY'.

The 68000 implementation of this transformation is shown in Listing One. To check the correctness of the program we must check the values of TARRAY for the test data (1 to 10) stored in ARRAY. If you run the program, the values stored will, of course, be 21, 44, 69, 96, and so on. Note that the problem did not specify the range of numbers that would be used in ARRAY. If our answers are to be correct then the result of the transformation (and the sum) should fit in a word length — that is, 16 bits.

This is, of course, by no means a unique solution to the problem, and there are other ways of achieving the same result. This obviously relates to the problem of program design and it is at this stage of the coding process that the importance of the subroutine — particularly in assembly language — becomes evident.

There are many methods of program design, but one way in which we can achieve what we might call a 'tidy' program (and one that will be easily comprehensible when we return to it later) is called *functional decomposition.* This is a method where the aim is to arrive at well-defined sections of programs that perform certain functions on data. The functions are described by verbs such as 'calculate', 'transpose' or 'check', and these sections of code, in a high-level language, are called *procedures*, with the data being passed via *parameters.*

At the level of assembler coding, the only convenient modular construction available to us is the subroutine, which on the 68000 is called by:

```
JSR   SUBR
```

This alters the program flow by causing a jump to the code section labelled SUBR. The code at SUBR will be executed until the instruction RTS is encountered, where the flow of program control then returns to the instruction following the calling JSR instruction (the mnemonic JSR stands for 'jump to subroutine', and RTS stands for 'return from subroutine').

Let's go back to our sample program. Suppose that instead of performing the calculation of squaring, multiplying and adding 'in one line', we wished to 'decompose' this into a subroutine call to achieve a better design. We would end up with something like the program shown in Listing Two. All that has changed is that on line 6 we have a subroutine call to CALC, and that what was in lines 6 to 9 is now contained in lines 12 to 15 — with an RTS placed at the end of this.

The advantage is that we now have a well-defined section of code (lines 12 to 16), performing a clearly defined operation on data passed to it via D1 (a parameter). Not only is this good design but also we can call this code — CALC — as many times as we like from wherever we like, in order to repeat the same calculation on any other data passed in D1 (and returned in D2)!

One further point to note when implementing subroutines, particularly in a multi-user environment, is that it is generally considered good practice for a subroutine to have only one entrance and one exit. This means that calls to other routines from within a subroutine should be avoided. It is also unwise to implement a large number of conditional 'returns', which can make debugging particularly difficult.

ADVANTAGES OF SUBROUTINES

The subroutine not only promotes good program design, but can also be thought of as a means whereby we can achieve an economical program (in terms of memory storage). This is because we would not now have to repeat the code for CALC each time we used it. It certainly is adaptable — both in terms of the ability to call CALC at many points in the program, and also because if we want to modify the transformation in some way we only have to look in one place in the program to do this.

For example, suppose the transformation should have been:

```
D2 := D1^2-20*D1
```

the new CALC subroutine has line 15 changed to:

```
SUB   D1,D2
```

in order to achieve this. Furthermore, not only is

Listing One

```
* Transformation of ARRAY into TARRAY using
* TARRAY[I] := ARRAY[I]2 + 20*ARRAY[I]
* Both arrays are 10 elements long
* Registers used are:
*     A0 points to ARRAY
*     A1 points to TARRAY
*     D0 is a loop counter
*     D1 is a temporary store
*     D2 is a copy of ARRAY[I]

          ORG     $1000
1 START   MOVEQ   #10,D0          set up loop count
2         LEA     ARRAY,A0        *set first pointer
3         LEA     TARRAY,A1       *and second
4         CLR     SUM             *make sure sum is zero
5 LOOP    MOVE    (A0)+,D1        *get ARRAY[I]
6         MOVE    D1,D2           *and copy
7         MULS    D2,D2           *form the square of ARRAY[I]
8         MULS    #20,D1          *D1 becomes 20 times ARRAY[I]
9         ADD     D1,D2           *add components
10        MOVE    D2,(A1)+        *store in TARRAY[I]
11        ADD     D2,SUM          *and keep a running total
12        SUBQ    #1,D0           *decrement loop count
13        BNE     LOOP            *repeat if not finished
14        TRAP    #0              *exit to monitor
15 ARRAY  DC.W    1,2,3,4,5,6,7,8,9,10
16 TARRAY DS.W    10
17 SUM    DS.W    1
18        END
```

Program Notes:

On lines 1 to 4, the variables used by the program are initialised; then each element of ARRAY is loaded into D1 using indirect addressing with post-increment, on line 5. A copy is made in D2 on line 6, so that we don't have to access the array again. Lines 7 and 8 form the two components of the transformation in D1 and D2, using the signed multiply instruction MULS, and line 9 adds these two together.

Line 10 stores the result in the correct element of TARRAY, and line 11 adds the result into the running sum, SUM. Finally, lines 12 and 13 are the familiar loop control statements so that 10 elements only are transformed — in effect, a FOR loop between lines 5 and 12/13, with the loop count being set up on line 1

Listing Two

```
          ORG     $1000
1 START   MOVEQ   #10,D0
2         LEA     ARRAY,A0
3         LEA     TARRAY,A1
4         CLR     SUM
5 LOOP    MOVE    (A0)+,D1
6         JSR     CALC            *call subroutine to calculate
7         MOVE    D2,(A1)+        *new array value
8         ADD     D2,SUM
9         SUBQ    #1,D0
10        BNE     LOOP
11        TRAP    #0
12 CALC   MOVE    D1,D2           *subroutine to calculate D1^2 + 20*D1
13        MULS    D2,D2           *and return the value in D2
14        MULS    #20,D1
15        ADD     D1,D2
16        RTS                     *return to the calling program
```

the program more adaptable but it's also easier to debug. For example, we can test CALC with all sorts of values passed to it in D1 and evaluate the results in D2 without any additional complications of 'embedded' programs or irrelevant data. There must only be one entry to the subroutine (at the subroutine address label), and only one exit (at the RTS instruction). Thus we can say that structuring our programs with subroutines will indeed make testing and debugging easier.

Overall, we seem to have satisfied our criteria for 'good programs' — at least in terms of economy, adaptability and reliability. Using subroutines does, however, entail a few drawbacks. In order to explore these — and also to examine the parameter passing — we need to look at the stack mechanism on the 68000.

THE STACK MECHANISM

When we execute a JSR instruction we need to remember the address of the next instruction in the calling sequence (the subroutine 'linkage'). The JSR instruction causes the address of the next instruction to be pushed onto the stack (using four bytes for the full long word address), and the program counter (PC) to be loaded with the address of the subroutine. When the RTS is executed in the subroutine, then the PC will be loaded with the address popped off the stack, and so execution continues after the subroutine call. All this address manipulation is transparent to the user because the 68000 takes care of everything. But there are side effects that the programmer should be wary of.

First of all, we must make sure that the stack pointer, SP, (or A7 on the 68000) is correctly set, otherwise we will overwrite code or data, or even violate hardware addresses! The usual way to do this is to use:

```
STACK   EQU   $1000        *set stack to 1000 hex
BEGIN   LEA   STACK, SP    *initialise the stack
                           pointer
```

and our stack will grow from 1000 down to zero (because each 'push' is a pre-decrement).

Secondly, we must make sure that the stack does not get too big! Here we have to estimate how much of the stack we are going to use — which can be very difficult. For relatively simple programs with simple subroutine calls and use of the stack, an examination of the code with a safety factor will give an accurate answer. For programs that are 'nested' (that is, where one subroutine calls another) or 'recursive' (where a subroutine calls itself), then the problem of stack sizing can be too difficult to evaluate, and other trial and error techniques have to be used.

In the next instalment, we will give more detailed examples of subroutine calls and, in particular, concentrate on methods of passing data to and from the subroutine, in some cases using some unique 68000 instructions.

A RARE BEAST

Besides being illegal, copying games software from one cassette to another cheats the authors and manufacturers out of thousands of pounds annually. Mikro-Gen's Shadow of the Unicorn incorporates 'dongle' technology, which the industry hopes will thwart the software pirates who have so far gone almost unchecked.

Dealing with tape piracy has become something of an obsession with software houses. The companies reckon they're losing thousands of pounds a year to pirates. The problem is that the most popular medium for computer games remains the cassette tape, which is, unfortunately for the software houses, the easiest to copy.

The idea of the 'dongle', a hardware device that plugs into the computer, has been around for some time. The dongle contains part of a program, and without it the rest of the program (held on cassette or disk) will not run. The problem with dongles is that they are more expensive to manufacture than standard cassettes. This means that software houses adopting the system must convince the public that dongles are in their own best interests and not just to benefit the manufacturer.

Mikro-Gen is the first of the games houses to adopt the dongle as a method of preventing software piracy. The first game to be launched under the new format is Shadow of the Unicorn, which is divided between a program held on a normal cassette tape and a dongle device that plugs into the Spectrum's expansion port.

The dongle itself consists of a 16-Kbyte EPROM, a joystick port and a joystick decoder chip. The EPROM contains the loading facilities for the cassette and a number of the graphics routines used throughout the game. On power up, the EPROM is banked into the areas of memory normally occupied by the BASIC ROM, allowing additional room for the remainder of the program.

An arcade-style adventure — rather in the style of Lords of Midnight (see page 495) — the game is played out in the kingdoms of Oronfal and Falforn, and the object is to defeat the forces of evil that now inhabit the land. Initially, the player controls three characters: Mithulin (the king of Oronfal), Ulin-Gail (a satyr) and Avarath (a wizard). There are, however, a number of other characters throughout the game who can be controlled by the player once they've been encountered. The first of these is Holdin, the captain of Falforn, who begins the game in the same position as the wizard and so immediately comes under player control.

As in other arcade adventures, there are not only a number of locations to explore, but also enemies to defeat and objects to collect that could help the character later in the game. In the early stages, the most common enemies are menacing and aggressive dwarf-like creatures, who can be easily destroyed by some of the characters, such as the wizard. Other characters can kill them only with great difficulty, while the remainder, who have no weapons at all, can only flee from a dwarf's attack.

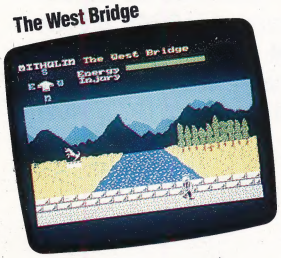
While you are controlling a particular character, your 'Energy' and 'Injury' factors are displayed as bars at the top of the screen. Obviously, a dwarf's attack will diminish the character's energy and if the dwarf is allowed to reach the character, the injury factor will rise. Energy can be replaced by feeding from one of the bushes distributed around the kingdom. Injuries, however, can generally be healed only with time.

As the characters move around the kingdom, they will come across a number of buildings that they might experience difficulty in entering. Attempting to move in the direction of the door will only cause the scene to change, leaving the character on the other side of the building. As you might expect, there are ways and means to enter the buildings, but only if they are approached in the right way.

In common with a number of other adventures, Shadow of the Unicorn is packaged with a book detailing the background to the game and providing the player with assorted hints on how to complete the adventure.

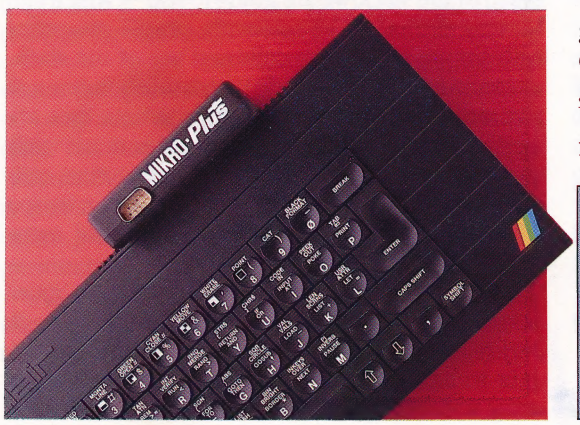
Shadow of the Unicorn is undoubtedly a gamble on the part of Mikro-Gen. But the company has clearly made an effort to present the game as attractively as possible to the customer. Whether it succeeds in redefining the market and finally eliminating piracy remains to be seen.

The West Bridge



Near Olindel

Walk This Way
Although Shadow of the Unicorn has a number of arcade elements included in its design, the basis of the game is that of a traditional adventure. The player controls a number of characters who search the various locations for clues and assistance in helping them defeat the powers of evil



Value Added Packs
The Mikro-Plus 'dongle', which is supplied with the Shadow of the Unicorn adventure, provides the player with an additional 16 Kbytes of usable memory, thus improving the extent and quality of the game. Also provided with the Mikro-Plus is a joystick interface, since Interface 2 cannot be fitted with the dongle

Shadow of the Unicorn: For the Sinclair Spectrum, £14.95
Publishers: Mikro-Gen, Unit 15, Western Centre, Bracknell, Berks
Author: Dale McLoughlin
Joystick: Optional
Format: Cassette and EPROM

Motorola 68000 Instruction Set

Here, courtesy of Motorola Inc, we present the first of three instalments in which we give details of the 68000's instruction set broken down into its eight component classes

Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Rx ⟷ Ry
LEA	32	EA → An
LINK	–	An → – (SP) SP → An SP + displacement → SP
MOVE	8, 16, 32	s → d
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA

Instruction	Operand Size	Operation
MOVEP	16, 32	(EA) → Dn Dn → (EA)
MOVEQ	8	#xxx → Dn
PEA	32	EA → – (SP)
SWAP	32	Dn[31:16] ⟷ Dn[15:0]
UNLK	–	An → Sp (SP) + → An

NOTES:
s = source d = destination – ( ) = indirect with predecrement
( ) + = indirect with postdecrement [ ] = bit number # = immediate data

Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	Dn + (EA) → Dn (EA) + Dn → (EA) (EA) + #xxx → (EA) An + (EA) → An
ADDX	8, 16, 32 16, 32	Dx + Dy + X → Dx – (Ax) + – (Ay) + X → (Ax)
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32 16, 32	Dn – (EA) (EA) – #xxx (Ax) + – (Ay) – An – (EA)
DIVS	32 ÷ 16	Dn ÷ (EA) → Dn
DIVU	32 ÷ 16	Dn ÷ (EA) → Dn
EXT	8 → 16 16 → 32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 × 16 → 32	Dn × (EA) → Dn
MULU	16 × 16 → 32	Dn × (EA) → Dn
NEG	8, 16, 32	0 – (EA) → (EA)
NEGX	8, 16, 32	0 – (EA) – X → (EA)
SUB	8, 16, 32 16, 32	Dn – (EA) → Dn (EA) – Dn → (EA) (EA) – #xxx → (EA) An – (EA) → An
SUBX	8, 16, 32	Dx – Dy – X → Dx – (Ax) – – (Ay) – X → (Ax)
TAS	8	[EA] – 0, 1 → EA[7]
TST	8, 16, 32	(EA) – 0

NOTES
[ ] = bit number
– ( ) = indirect with predecrement
( ) + = indirect with postincrement
= immediate data

© 1983 MAGNENAT-THALMANN, THALMANN—
UNIV. OF MONTREAL